David Zhan, Landrick Diaz,
Angela Ho, Laurie Lee

Milestone 01

Team 21

Overview & Members

Project Name: Streetside
Team Name: Sustainable Squirrels (Team 21)
Date: Friday, April 4, 2025
Repo: https://github.com/lhlee-umass/streetside.git

Project Summary

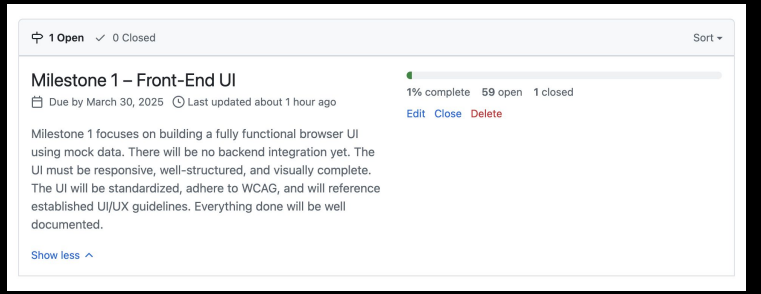
Due to overconsumption and impulse buying, many have products they don't want or need anymore. In this milestone, we built the functional front end of an app that allows users to browse items up for sale, message the seller if interested, and check out the seller's profile for reviews. We included essential elements to the app like the homepage, user page, messaging page, navigation between pages, and integrated mock data of listings, users, and reviews using IndexedDB & TS structures.

Laurie Lee focused primarily on developing reusable components for the application: developing the user profile component and layout, messaging bubble component and messaging interface, and layout of the listing cards on the homepage and ensuring that the filtering functionality worked.

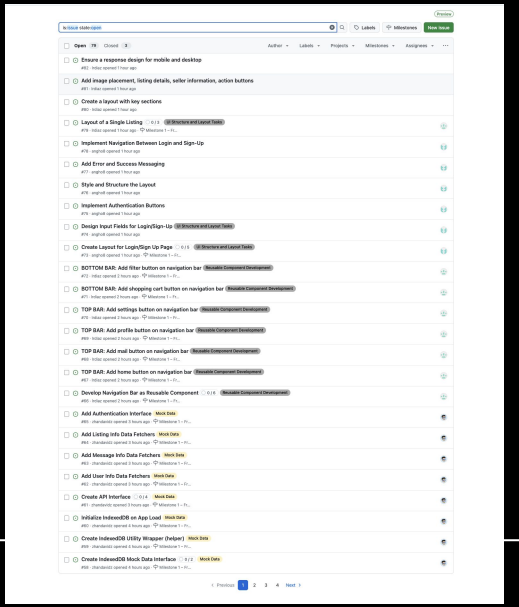
Landrick Diaz focused on creating two main components, such as the listing card component and the navigation bar.

Angela Ho focused primarily on the the UI style developing the filtering buttons, the messaging listing page and login/signup.

David Zhan focused

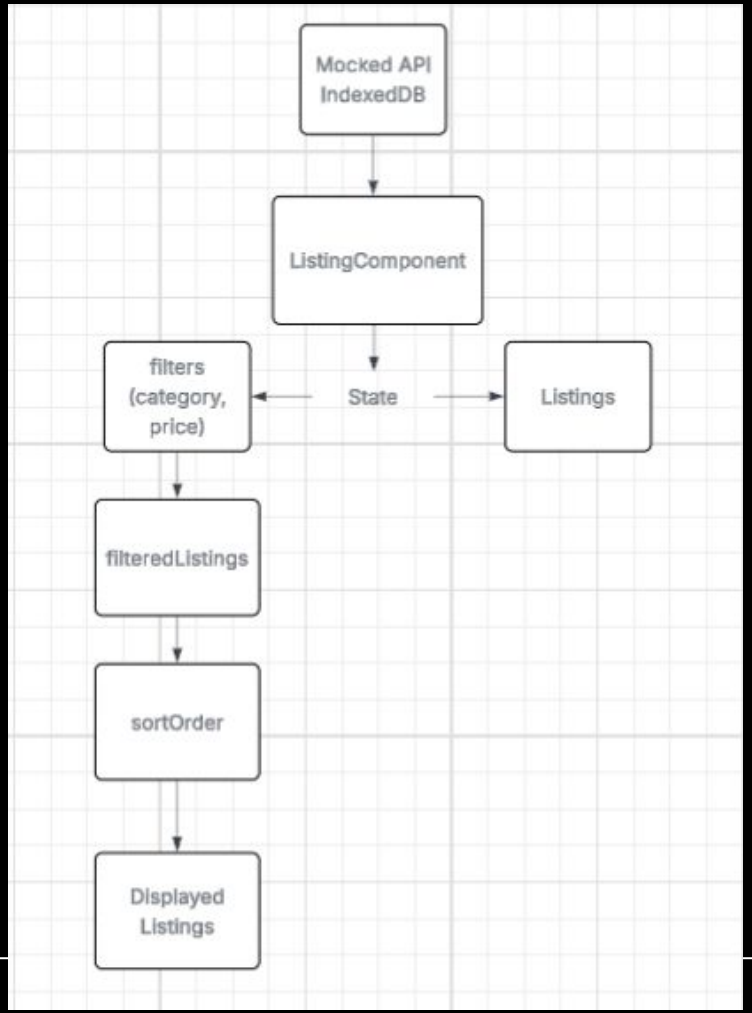


We had about 78 issues and sub-issues, this is one of four pages of issues that is in our GitHub

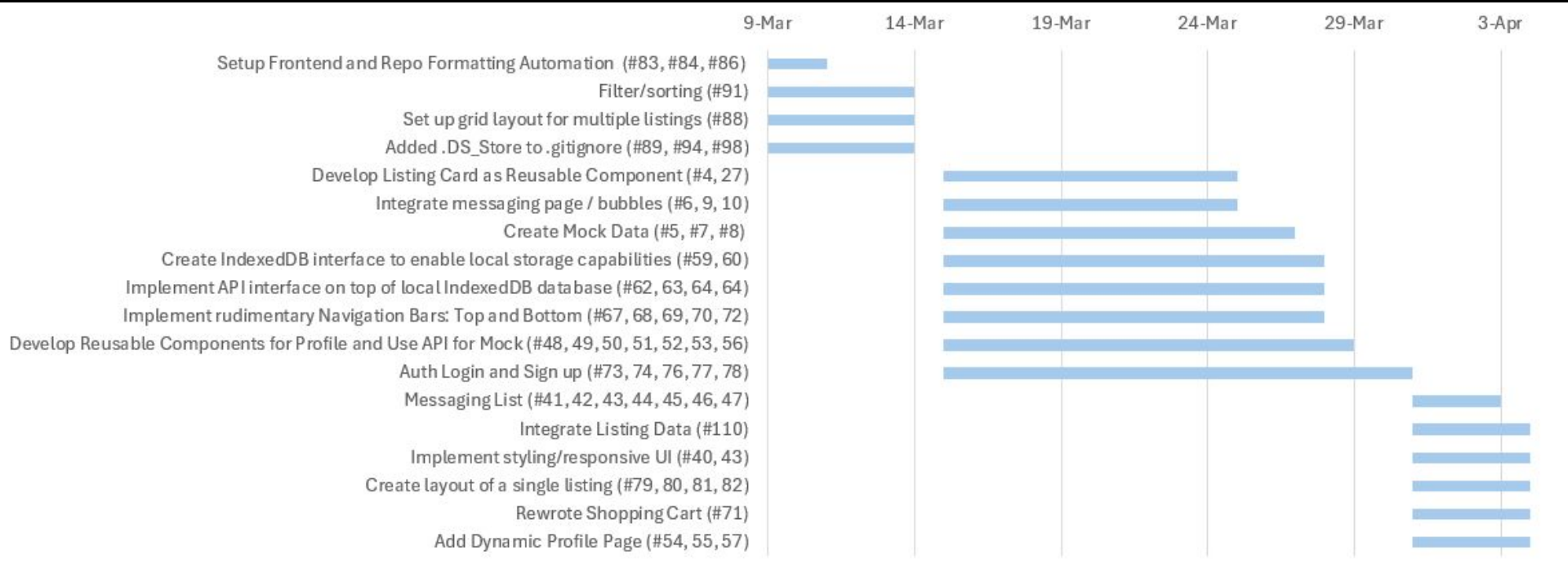


Software Architecture

- The ListingComponent fetches the list of all items (regular listings) from the mock API (IndexedDB) and manages the state
- State management: The ListingComponent holds three states:
    - The filters stores the user's selected filters (search term, category, price, condition).
    - The filtered listings stores the listings after filters are applied.
    - sortOrder stores the user's sorting preference
- When a user interacts with the Filters (e.g., selecting a category, entering a search term), the filteredListings state is updated to show only the listings that match the filter criteria.
- Once the Filters are applied, the sortOrder state is used to sort the filteredListings (e.g., by price low-to-high, newest, or distance).
- Data flow: The final filtered and sorted listings are passed to the Listing Display to be shown in the UI.



Historical Development Timeline



Design and Styling Guidelines

MAIN POINTS

Used **Tailwind CSS** for formatting

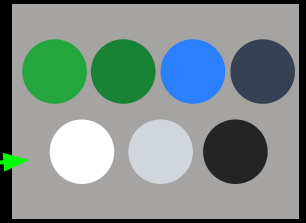
Color Schemes: reflects a modern, utility look with some splashes of blue and green buttons to encourage users to execute actions (e.g., button for Message Seller).

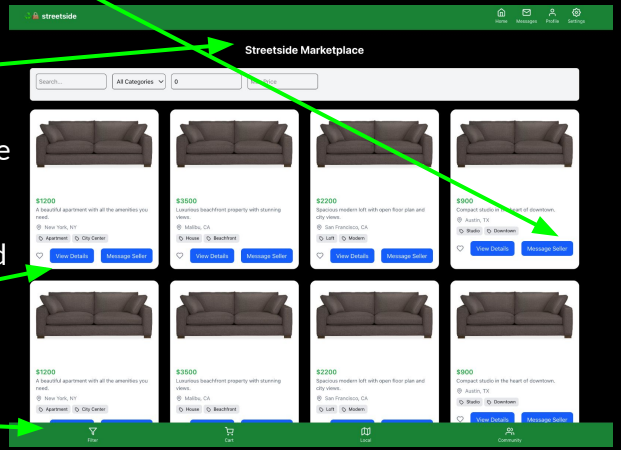
Typography: Type styles should be one of the five following roles: Display, headline, title, body, and label, as Google's Material Design for Typography suggests (to highlight what is most relevant)

Layout: Layout should direct attention to the action users want to make and adapted to diff size windows as Google's Material Design for Layout basics suggests

Accessibility: All text should have high contrast with its background, anticipating, including and responding the needs of individuals, as WCAG Accessibility Guidelines suggests (makes text readable)

Responsive Design: Navigation bar should use 100% of the screen width in compact window sizes, as Google's Material Design for Responsive Layout suggests



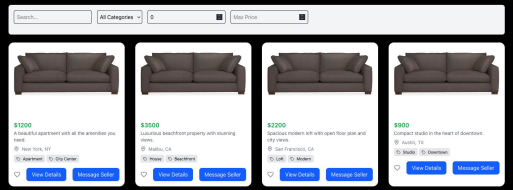


Link to Full UI/UX Design and Style Guidelines Document

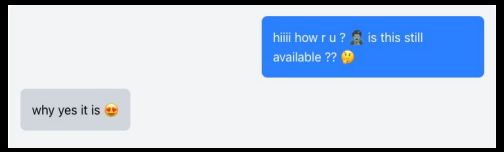
Component Documentation

ListingCard: This component displays a preview of an item listing in a card format, including key details such as the image, price, title, location, and category tags. It provides optional action buttons that allow users to favorite a listing or message the seller. It accepts props including id, title, image, price, description, location, tags[], as well as callback functions for user interaction like onFavorite() and onMessageSeller(). The component is responsive and designed for use within grid-based layouts on pages like the homepage or search results.

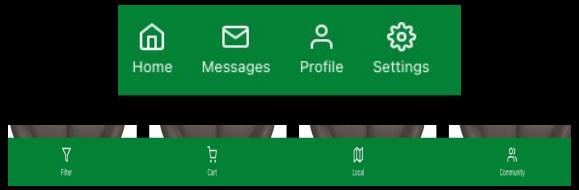
It has a filter-sorting sub-component that provides filters for search, category, condition, and price range. It accepts a list of listings[] as props and includes an interactive UI for narrowing down results. As users enter values or select options, the listing results dynamically update in real-time using useState and useEffect. Sorting options are also available, allowing users to sort by price, date, or distance without requiring manual submission, creating a smooth and responsive filtering experience.



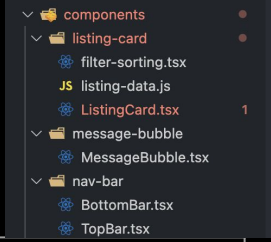
MessageBubble: This component renders a single message bubble in a chat interface and visually distinguishes between messages sent by the user and those sent by the seller. It accepts two props: sender, which determines the alignment and style of the message ('user' or 'seller'), and text, which contains the message content. The component uses conditional Tailwind CSS classes to align bubbles to the left or right and apply distinct background and text colors based on the sender. It is designed for use in messaging views to create a clear and intuitive conversation layout.



TopBar & BottomBar: These two components define the fixed navigation bars at the top and bottom of the app interface. TopBar provides quick access to core app sections such as Home, Messages, Profile, and Settings, while also including branded app text for identity. BottomBar is designed to help users navigate location-based and community features, including Filters, Cart, Local Map, and Community. Both components use icon-based links (react-icons/fi) combined with text labels for clarity, and they feature responsive styling with Tailwind classes for layout, spacing, and hover states. Each Link points to a corresponding route, enabling smooth client-side navigation across the app.

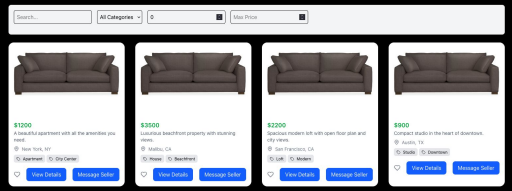


Component Documentation

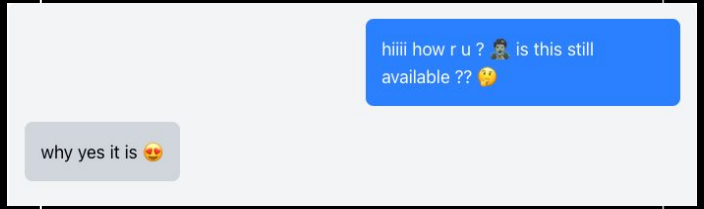


ListingCard: This component displays a preview of an item listing in a card format, including key details such as the image, price, title, location, and category tags. It provides optional action buttons that allow users to favorite a listing or message the seller. It accepts props including id, title, image, price, description, location, tags[], as well as callback functions for user interaction like onFavorite() and onMessageSeller(). The component is responsive and designed for use within grid-based layouts on pages like the homepage or search results.

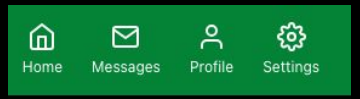
It has a filter-sorting sub-component that provides filters for search, category, condition, and price range. It accepts a list of listings[] as props and includes an interactive UI for narrowing down results. As users enter values or select options, the listing results dynamically update in real-time using useState and useEffect. Sorting options are also available, allowing users to sort by price, date, or distance without requiring manual submission, creating a smooth and responsive filtering experience.



MessageBubble: This component renders a single message bubble in a chat interface and visually distinguishes between messages sent by the user and those sent by the seller. It accepts two props: sender, which determines the alignment and style of the message ('user' or 'seller'), and text, which contains the message content. The component uses conditional Tailwind CSS classes to align bubbles to the left or right and apply distinct background and text colors based on the sender. It is designed for use in messaging views to create a clear and intuitive conversation layout.

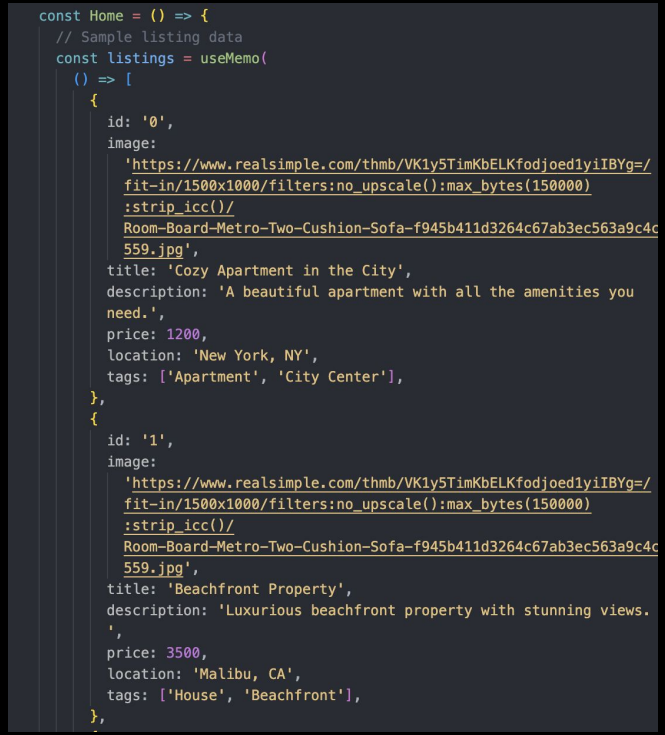


TopBar & BottomBar: These two components define the fixed navigation bars at the top and bottom of the app interface. TopBar provides quick access to core app sections such as Home, Messages, Profile, and Settings, while also including branded app text for identity. BottomBar is designed to help users navigate location-based and community features, including Filters, Cart, Local Map, and Community. Both components use icon-based links (react-icons/fi) combined with text labels for clarity, and they feature responsive styling with Tailwind classes for layout, spacing, and hover states. Each Link points to a corresponding route, enabling smooth client-side navigation across the app.



Performance Considerations

We used useMemo in Home.tsx to memoize the static listings[] array and prevent it from being re-created on every render. This ensures that the component only computes the listing data once, improving performance especially during frequent UI updates like filtering or sorting. By maintaining a stable reference to the listings array, we reduce unnecessary re-renders of child components like ListingCard. While this optimization is minor for static data, it lays the groundwork for handling larger datasets or dynamic fetches in the future. For even greater efficiency, useMemo can also be applied to filter and sort operations when those depend on user input or application state.



```
const Home = () => {
  // Sample listing data
  const listings = useMemo(
    () => [
      {
        id: '0',
        image:
          'https://www.realsimple.com/thmb/VK1y5TimKbELKfodjoed1yiIBYg=/
          fit-in/1500x1000/filters:no_upscale():max_bytes(150000)
          :strip_icc()/
          Room-Board-Metro-Two-Cushion-Sofa-f945b411d3264c67ab3ec563a9c4c
          559.jpg',
        title: 'Cozy Apartment in the City',
        description: 'A beautiful apartment with all the amenities you
        need.',
        price: 1200,
        location: 'New York, NY',
        tags: ['Apartment', 'City Center'],
      },
      {
        id: '1',
        image:
          'https://www.realsimple.com/thmb/VK1y5TimKbELKfodjoed1yiIBYg=/
          fit-in/1500x1000/filters:no_upscale():max_bytes(150000)
          :strip_icc()/
          Room-Board-Metro-Two-Cushion-Sofa-f945b411d3264c67ab3ec563a9c4c
          559.jpg',
        title: 'Beachfront Property',
        description: 'Luxurious beachfront property with stunning views.
        ',
        price: 3500,
        location: 'Malibu, CA',
        tags: ['House', 'Beachfront'],
      },
```

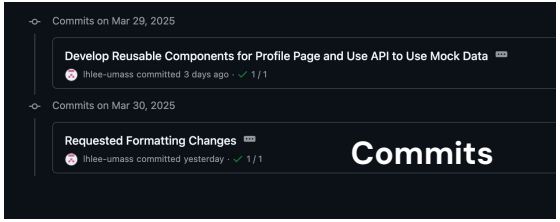
Laurie Lee (1/3)

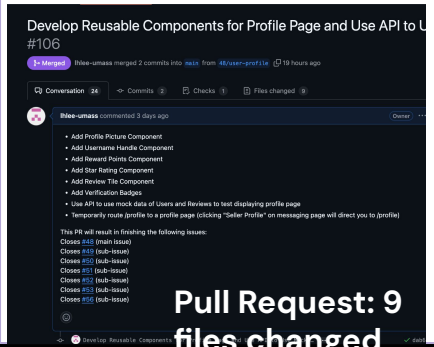
**No issue assigned to Laurie for Milestone 1 was left unclosed.

Full Name: Laurie Lee
UMass email: lhlee@umass.edu
GitHub ID: lhlee-umass
Assigned Work Summary: Most of my work involved creating reusable components for the application with mock data.
LINK TO ALL COMPLETED/CLOSED ISSUES BY LAURIE LEE

User Profile (buyer/seller) & Layout

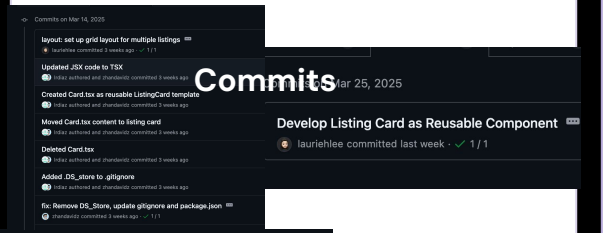
Develop User Profile Main Issue, which also completes subissues #49,50,51,52,53,56
Associated Pull Request





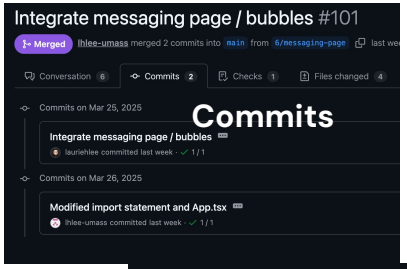
Listing Card Layout

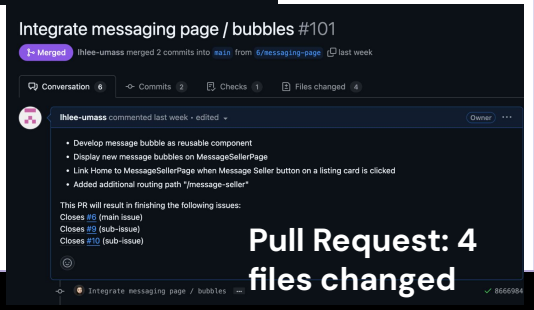
Develop Listing Card Main Issue, which also completes subissues #21,22,24,26,27
Setup grid layout for multiple listings
Ensure listing page functionality
Associated: PR-Listing , PR-Layout



Messaging Bubble & Interface Layout

Develop Messaging Bubbles as Reusable Component, which also completes subissues #9,10
Associated Pull Request



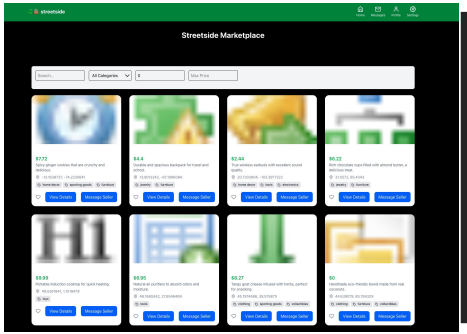


Laurie Lee (2/3)

Full Name: Laurie Lee
UMass email: lhlee@umass.edu
GitHub ID: lhlee-umass
Code & UI Explanation

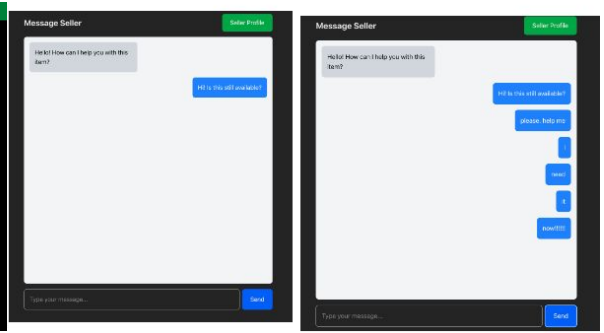
Let's first look at the flow of pages I created and linked to one another. On the next slide, we'll cover details regarding each page. When a user wants to look for a listing item on Streetside, they first check out the homepage. If they're interested in an item, they can message the seller. They can check out the seller profile to see if they're reliable.

Homepage of all the listings



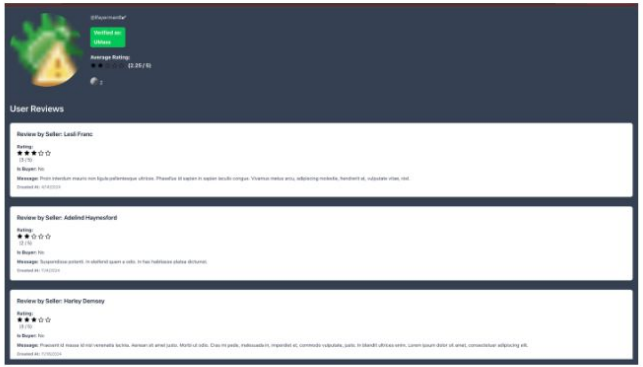
Click on "Message Seller" on a listing card to move to the Messaging interface.

Messaging Interface



Write a message in the input box and press send to view the sent messages.

User Profile (buyer/seller)



Click on "Seller Profile" on the Messaging Interface to move to the user profile.

Laurie Lee (2/3)

Full Name: Laurie Lee
UMass email: lhlee@umass.edu
GitHub ID: lhlee-umass
Code & UI Explanation

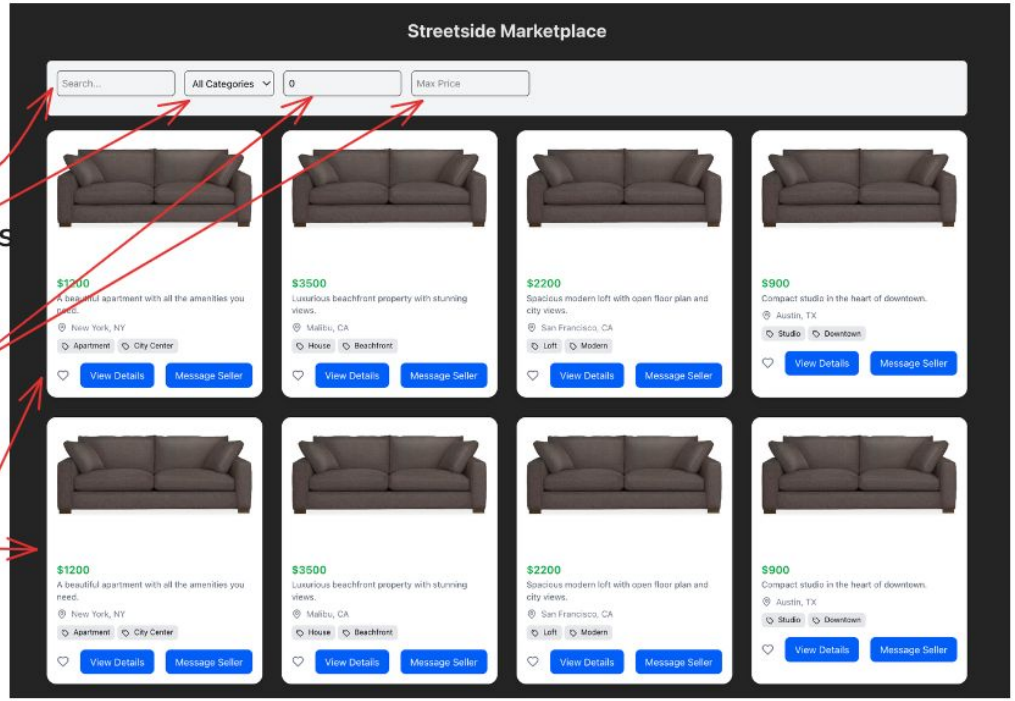
```
// State for filtered listings
const [filteredListings, setFilteredListings] = useState(listings)

// Effect hook to apply filters
useEffect(() => {
  let updatedListings = [...listings]

  // Search filter
  if (filters.searchTerm) {
    updatedListings = updatedListings.filter((listing) =>
      listing.title.toLowerCase().includes(filters.searchTerm.toLowerCase())
    )
  }

  // Category filter
  if (filters.category) {
    updatedListings = updatedListings.filter((listing) =>
      listing.tags.includes(filters.category)
    )
  }

  // Price filter
  updatedListings = updatedListings.filter(
    (listing) =>
      listing.price >= filters.minPrice && listing.price <= filters.maxPrice
  )

  setFilteredListings(updatedListings)
}, [filters, listings])
```

Ensured functionality of filtering ability:
- Search bar which allows user to sort by tag
- Categories which displays all possible tags
- Min and max price filters out the listings that are outside of the range

```
      </div>

      {/* Grid layout for listings */}
      <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:
        {filteredListings.map((listing, index) => (
          <ListingCard
            key={index}
            image={listing.image}
            title={listing.title}
            description={listing.description}
            price={listing.price}
            location={listing.location}
            tags={listing.tags}
            onFavorite={() => {
              console.log(`Favorite clicked for ${listing.title}`)
            }}
            onMessageSeller={() => {}}
          />
        ))}
      </div>
    </div>
  </>
```

Implemented Grid Layout of Listing for homepage

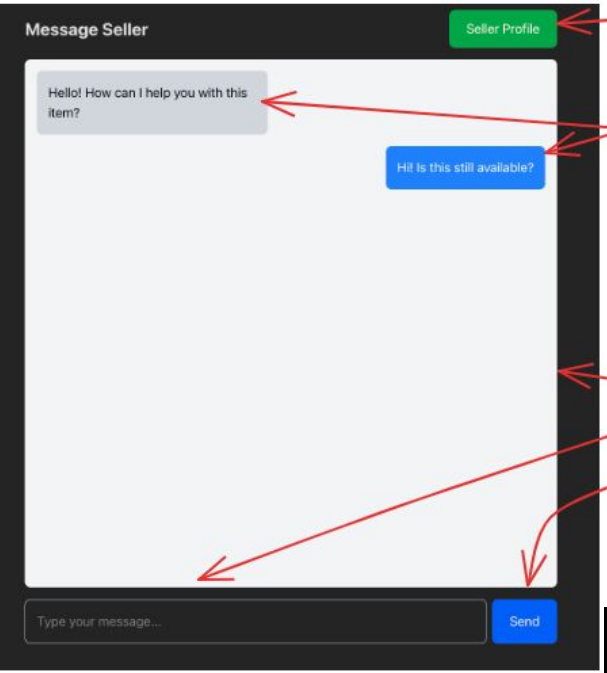
The Homepage



Laurie Lee (2/3)

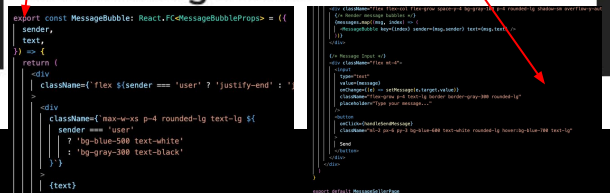
Full Name: Laurie Lee
UMass email: lhlee@umass.edu
GitHub ID: lhlee-umass
Code & UI Explanation

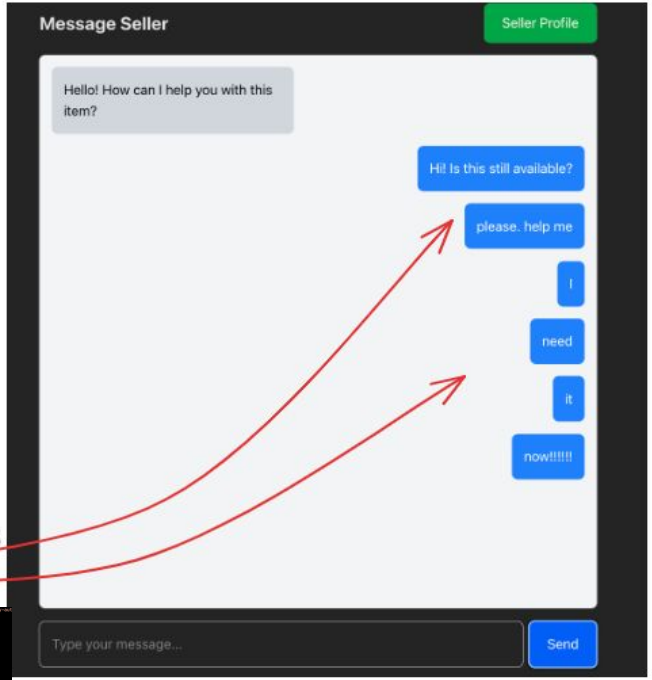
The Messaging Interface



- Seller Profile button, which links to profile page
- Messaging Bubble as a reusable component: blue for sender, gray for received message

- Box containing messages
- Input box for text to send
- Send button
- Once message is sent, appears in message box



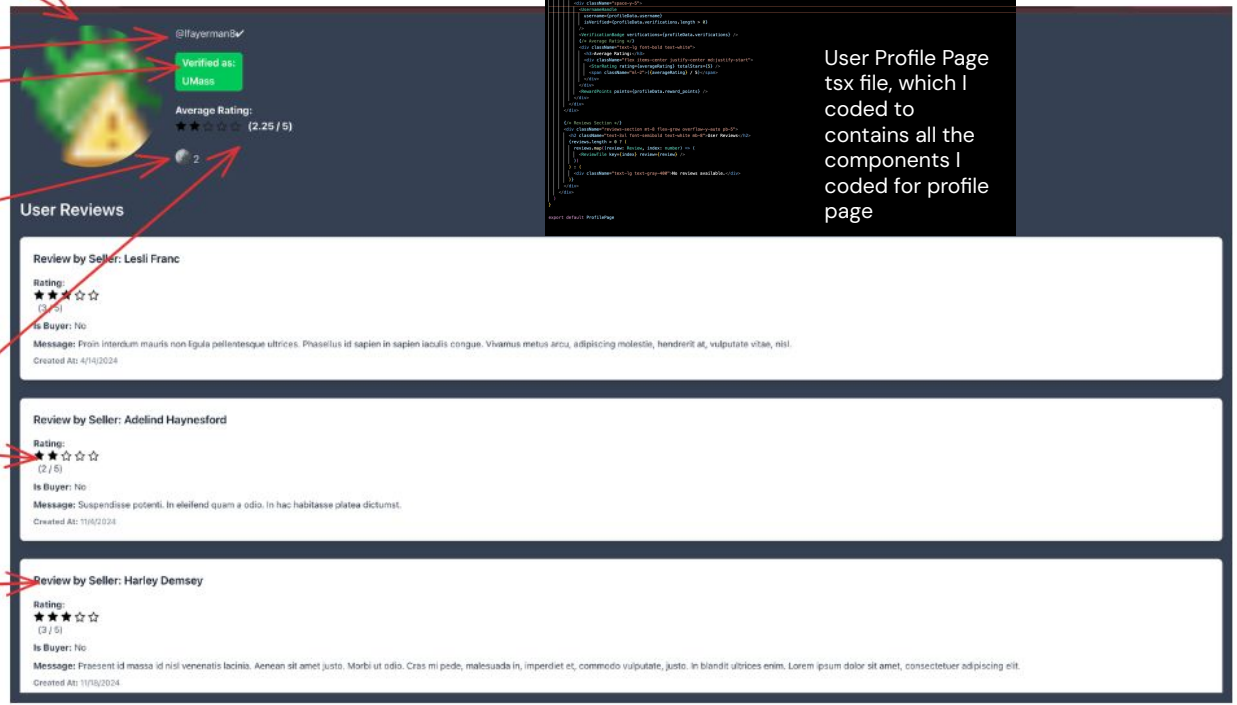


Laurie Lee (2/3)

Full Name: Laurie Lee
UMass email: lhlee@umass.edu
GitHub ID: lhlee-umass
Code & UI Explanation

- Profile Picture Component
- Username Handle Component
- Verification Badges (shows how user is verified, e.g., from UMass)
- Reward Points Component (displays gold coin representing points)
- Star Rating Component (average rating for user at the top, individual ratings per review)
- Review Tile Component (includes star rating component and clarifies if review is from seller or buyer)
- Use David's API for profile page data (user info & reviews info)

API built on IndexedDB



User Profile Page tsx file, which I coded to contains all the components I coded for profile page

```
// Get reviewer user data
useEffect(() => {
  const fetchData = async () => {
    try {
      const userData = await Users.getUser(reviewer_id)
      setUserData(userData)
    } catch (error) {
      console.error('Failed to fetch data:', error)
    }
  }

  fetchData()
}, [reviewer_id])
```

Review tile includes API call to David's mock data

Laurie Lee (2/3)

Full Name: Laurie Lee
UMass email: lhlee@umass.edu
GitHub ID: lhlee-umass
Code & UI Explanation (Component Hierarchy & Interaction)

The previous slides outlined the pages / components I worked on.

Interaction w/ Angela's components:
After a user signs up/logs in (implemented by Angela), they can view the listings, message a user, and check out a user profile if they're interested in checking out who posted a listing. The messaging user page I created is a **direct message to a seller**, which interacts with the messaging list page because messaging a seller will become a conversation in the messaging list page, which Angela implemented.

Interaction w/ Landy's components:
Generally navigating between these pages can be done by the navigation bar on the home page/marketplace page (which Landy implemented, which he will outline in his slides). Navigation can also be done through buttons– I outlined this particular **user flow representation on slide 9**, which I implemented.

Interaction w/ David's components:
The listings, messages, and users rely on on mock data generated by David. The mock data is obtained by calling an API built on IndexedDB generated by David (there is no backend service).

Laurie Lee (3/3)

Full Name: Laurie Lee
UMass email: lhlee@umass.edu
GitHub ID: lhlee-umass
Challenges & Insights

Key Takeaways

Overall, I had a great time completing Milestone 1 because our team collaborates very well. We meet and communicate often, and resolve issues quickly with ease. When we have different opinions or thoughts, it's never blocks our general progress.

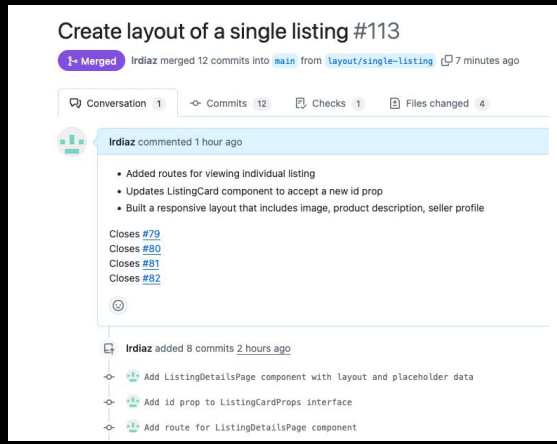
Obstacles faced & Solutions

An obstacle that I faced was trying to figure out how to wrap my head around Tailwind CSS because I was used to using separate .css files for frontend development. I thought that it might look messy using Tailwind CSS but it actually streamlines everything so much better in the general project file structure. Once I got used to the additional formatting text in the TS files overtime, I was able to filter out the noise and identify where components were coded.

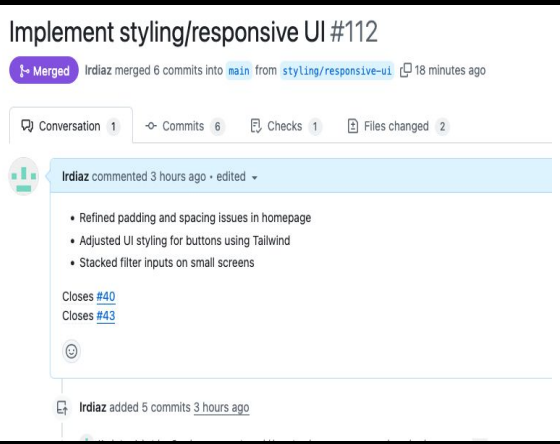
Building off of understanding new concepts / familiarizing yourself with them overtime, I've learned through this milestone that it's important to do things gradually, because before I knew it, I felt a lot more comfortable with front end development than I used to. My front end development experience was extremely limited before this project, so now, I feel more confident in my abilities. I also get the general feeling that I completed a lot more than I had thought I could complete at the beginning of the milestone.

Landrick Diaz (1/3)

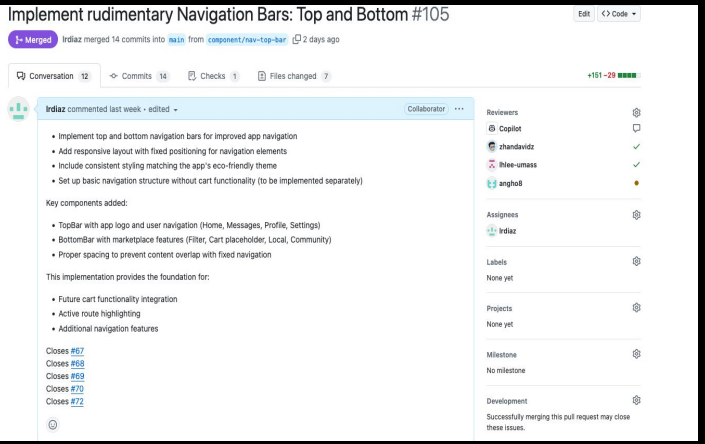
Full Name: Landrick Diaz
UMass email: lrdiaz@umass.edu
GitHub ID: lrdiaz



Creating Layout of a Single Listing
(A larger view of grid listing)
Issues #79, 80, 81, 82
Pull Request #113



Implement styling/responsive UI
(Making sure browser adjusts UI depending on screen size)
Issues #40, 43



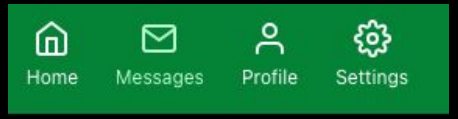
Implementing Rudimentary Nav. Bars
(Top and Bottom Bars, placeholder)
Issues #67, 68, 69, 70, (71), 72

Landrick Diaz (2/3)

Full Name: Landrick Diaz
UMass email: lrdiaz@umass.edu
GitHub ID: lrdiaz

```tsx
export default function BottomBar() {
  return (
    <nav className="fixed bottom-0 left-0 right-0 bg-green-700 text-white
    p-3 shadow-lg z-10">
      <div className="container mx-auto flex justify-around items-center">
        <Link
          to="/filter"
          className="flex flex-col items-center hover:text-green-200"
        >
          <FiFilter className="w-6 h-6" />
          <span className="text-xs mt-1">Filter</span>
        </Link>
        <Link
          to="/cart"
          className="flex flex-col items-center hover:text-green-200"
        >
          <FiShoppingCart className="w-6 h-6" />
          <span className="text-xs mt-1">Shopping Cart</span>
        </Link>
        <Link
          to="/map"
          className="flex flex-col items-center hover:text-green-200"
        >
          <FiMap className="w-6 h-6" />
          <span className="text-xs mt-1">Local</span>
        </Link>
        <Link
          to="/community"
          className="flex flex-col items-center hover:text-green-200"
        >
          <FiUsers className="w-6 h-6" />
          <span className="text-xs mt-1">Community</span>
        </Link>
      </div>
    </nav>
  );      You, last week • Implemented BottomBar.tsx …
}
```

```tsx
<nav className="fixed top-0 left-0 right-0 bg-green-700 text-white p-4
shadow-md z-10">
  <div className="container mx-auto flex justify-between items-center">
    <Link
      to="/"
      className="text-xl font-bold flex items-center
      hover:text-green-200"
    >
      <span className="mr-2">🚲</span>
      streetside
    </Link>
    <div className="flex space-x-6">
      <Link
        to="/"
        className="flex flex-col items-center hover:text-green-200"
      >
        <FiHome className="w-6 h-6" />
        <span className="text-xs mt-1">Home</span>
      </Link>
      <Link
        to="/messages"
        className="flex flex-col items-center hover:text-green-200"
      >
        <FiMail className="w-6 h-6" />
        <span className="text-xs mt-1">Messages</span>      You, last w
      </Link>
      <Link
        to="/profile"
        className="flex flex-col items-center hover:text-green-200"
      >
        <FiUser className="w-6 h-6" />
        <span className="text-xs mt-1">Profile</span>
      </Link>
      <Link
        to="/settings"
        className="flex flex-col items-center hover:text-green-200"
      >
        <FiSettings className="w-6 h-6" />
        <span className="text-xs mt-1">Settings</span>
      </Link>
```



Each of the options in the top and bottom navigation bar is linked to a path using the React Router. Upon clicking on one of the option, you will be taken to a different page.

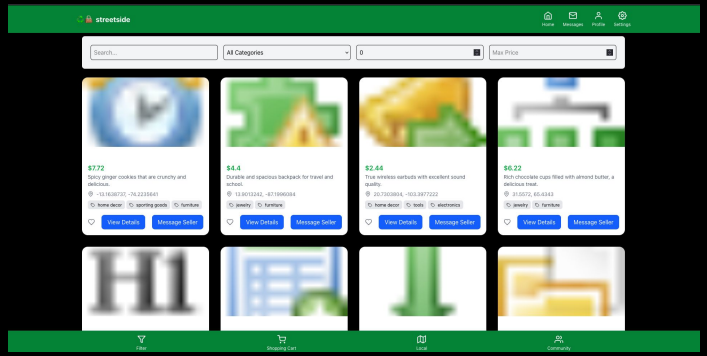
Landrick Diaz (2/3)

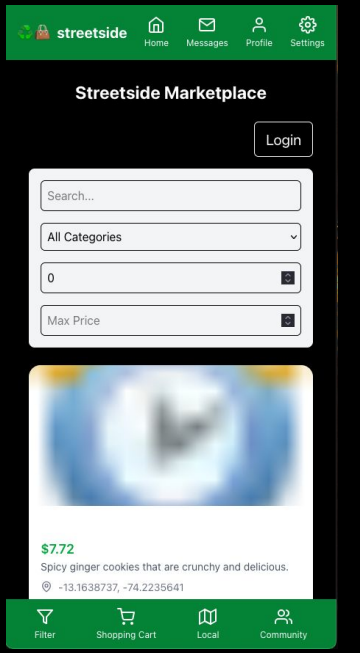
Full Name: Landrick Diaz
UMass email: lrdiaz@umass.edu
GitHub ID: lrdiaz

```tsx
export default function App() {
  return (
    <div className="min-h-screen flex flex-col m-0 p-0 text-[#213547]
    bg-white dark:text-white dark:bg-black">
      <TopBar />
      <main className="flex-grow pt-20 pb-16 px-4">
        <Outlet />
      </main>
      <BottomBar />
    </div>
  )
}
```

```tsx
            {' '}
            <Heart size={20} />
          </button>
        ))}
        <Link to={`/listing/${id}`} className="w-full md:w-auto">
          <button className="w-full md:w-auto px-4 py-2 bg-blue-600
          text-white rounded-lg hover:bg-blue-700">
            View Details
          </button>
        </Link>

        {/* Message Seller Button */}
        {onMessageSeller && (
          <Link to="/message-seller">
            <button className="w-full md:w-auto px-4 py-2 bg-blue-600
            text-white rounded-lg hover:bg-blue-700">
              Message Seller
            </button>
          </Link>
        )}
      </div>
    </CardContent>
```

Using TailwindCSS allowed me to implement Responsive UI Styling. Initially, the website did not adapt well to having smaller screens — I would often play with the size of my browser to make sure that what I was seeing looked proper. By adding TailwindCSS styles into a bunch of various TSX elements, I was able to have the website adapt to smaller screens. As you ccan see on the right, the listings become stacked vertically when it is in a “mobile”-sized screen

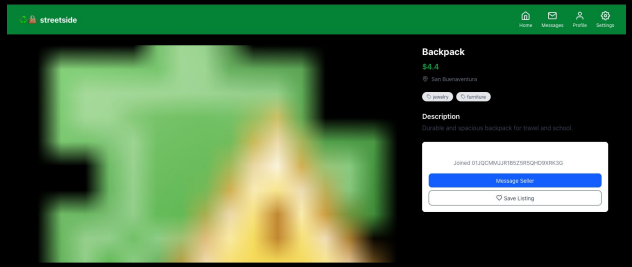




Landrick Diaz (2/3)

Full Name: Landrick Diaz
UMass email: lrdiaz@umass.edu
GitHub ID: lrdiaz

```jsx
// Handle loading and error states      Laurie Lee, 2 hours ago • Make compatible with listing data from
if (isLoading)
  return <p className="text-center text-gray-500 mt-10">Loading...</p>
if (error) return <p className="text-center text-red-500 mt-10">{error}</p>
if (!listing)
  return <p className="text-center text-gray-500 mt-10">Listing not found.</p>

return (
  <div className="container mx-auto px-4 py-6">
    {/* Desktop grid, stacked on mobile */}
    <div className="grid grid-cols-1 lg:grid-cols-3 gap-8">
      {/* Left side: Images */}
      <div className="lg:col-span-2">
        {/* Loop through images array */}
        {listing.images && listing.images.length > 0 ? (
          <img
            src={listing.images[0]} // Display the first image for now (you can extend this for a
            gallery)
            alt={listing.title}
            className="w-full aspect-video object-cover rounded-xl shadow"
          />
        ) : (
          <p>No images available</p>
        )}
      </div>
```




```jsx
      {/* Right side: Details */}
      <div className="flex flex-col gap-6">
        {/* Title, price, location */}
        <div className="flex flex-col gap-2">
          <h1 className="text-2xl font-bold">{listing.title}</h1>
          <p className="text-green-600 text-xl font-semibold">
            ${listing.price}
          </p>
          <div className="flex items-center gap-2 text-sm text-gray-500">
            <MapPin size={16} />
            <span>{listing.location_name}</span> {/* Display location name */}
          </div>
        </div>

        {/* Tags */}
        <div className="flex flex-wrap gap-2">
          {listing.tags.map((tag: string, index: number) => (
            <span
              key={index}
              className="flex items-center gap-1 bg-gray-200 text-gray-700 text-xs px-3 py-1
              rounded-full"
            >
              <Tag size={12} />
              {tag}
            </span>
          ))}
        </div>
```

```jsx
        {/* Description */}
        <div>
          <h2 className="text-lg font-semibold mb-1">Description</h2>
          <p className="text-gray-700">{listing.description}</p>
        </div>

        {/* Seller Info */}
        <div className="border rounded-lg p-4 bg-white shadow-sm">
          <div className="flex items-center gap-4">
            <img
              src={listing.seller_id} // Assuming 'seller' object exists
              alt={listing.seller_id} // Assuming 'seller' object exists
              className="w-12 h-12 rounded-full object-cover"
            />
            <div>
              <p className="font-semibold">{listing.seller_id}</p>
              <p className="text-sm text-gray-500">
                Joined {listing.seller_id}{' '}
                {/* Assuming seller's 'joined' date */}
              </p>
            </div>
          </div>
          <Link to="/message-seller">
            <button className="w-full mt-4 px-4 py-2 bg-blue-600 text-white rounded-lg
            hover:bg-blue-700 text-sm">
              Message Seller
            </button>
          </Link>
          <button className="w-full mt-2 px-4 py-2 border text-gray-700 rounded-lg hover:bg-gray-100
          text-sm flex justify-center items-center gap-1">
            <Heart size={16} /> Save Listing
          </button>
```

In ListingDetailsPage, I start by grabbing the id from the URL using useParams() — this tells me which listing the user clicked on. I use useEffect to run a function when the component loads, which fetches that specific listing using Listings.getListing(id) from our mock IndexedDB API that David built. While it's loading, I show a "Loading..." message, and if something goes wrong (like a missing or broken ID), I show an error message or a fallback. Once the listing is successfully fetched, I store it in state and display it using a responsive grid: the image shows on the left (or top on mobile), and the details like title, price, location, and tags go on the right. There's also a description section and a seller info box — although right now I'm using seller_id as a placeholder for the seller's avatar and name, which I plan to improve later. At the bottom, I include buttons to message the seller or save the listing, tying the whole single-listing experience together in a way that feels clear and familiar, like Facebook Marketplace.

Landrick Diaz (2/3)

Full Name: Landrick Diaz
UMass email: lrdiaz@umass.edu
GitHub ID: lrdiaz

Interaction w/ Angela's components:
When you click on the Messages tab in the navigation bar (which I built), it brings up the messaging UI that Angela worked on. I also connect to her components in the item listing cards — each one has a "Message Seller" button that lets you start a new conversation right from that item. So there are two ways to access the messaging system: one to continue an existing chat, and one to start a new one directly from a listing.

Interaction w/ Laurie's components
In the bottom navigation bar, I added a Filter tab that links to the filtering UI Laurie made. Her component lets users filter by category, price range, and keywords, and I just made sure there was an easy way for users to get there. It works well with the homepage grid, and it helps make the listing results feel more tailored and usable.

Interaction w/ David's components
David's mock data (from his IndexedDB API) powers most of the stuff I worked on visually. For example, the single listing page I built pulls in the actual data from his mock backend. It made it a lot easier to see how things would work in the real app, instead of just guessing or hardcoding values.

Landrick Diaz (3/3)

Full Name: Landrick Diaz
UMass email: lrdiaz@umass.edu
GitHub ID: lrdiaz

Milestone 1 was honestly more fun than I expected. I felt that our team luckily had a great rhythm. We would talk often and keep things moving. That made a big difference for me, especially because a lot of this tech was new to me. It felt good to build something that actually looks and feels real. Seeing the app come together and knowing I had a hand in that, even just small parts like the ListingCard or the navigation bar, made it feel more tangible than past projects.

Obstacles Faced & Solutions

One of the biggest hurdles for me was getting used to Tailwind CSS. I've always written styles in separate .css files, so seeing all these utility classes packed into a component felt messy at first. It still is messy to me, but I appreciate how I don't have to jump from one file to another trying to format a TSX element. Another thing I didn't expect to struggle with was using Git *with a team*. This is especially ironic as a UCA for 326, where we teach this in the first few labs. Adding on top of that all the JSON and NPM stuff, it was a little chaotic. It's especially important to create branches and make sure that people review your code so you don't mess up anything grand. I'd say that I still have a lot to learn about how all these things (React, TS, routing) interact with each other, both conceptually and written, but I feel that I have made progress.

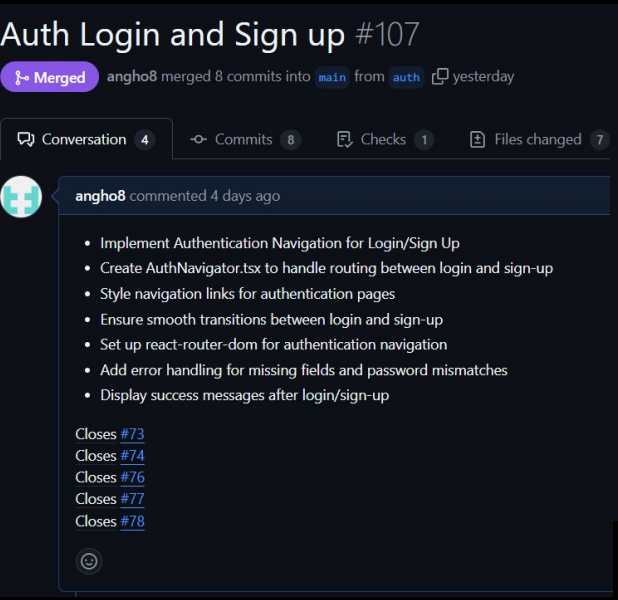
Angela Ho (1/3)

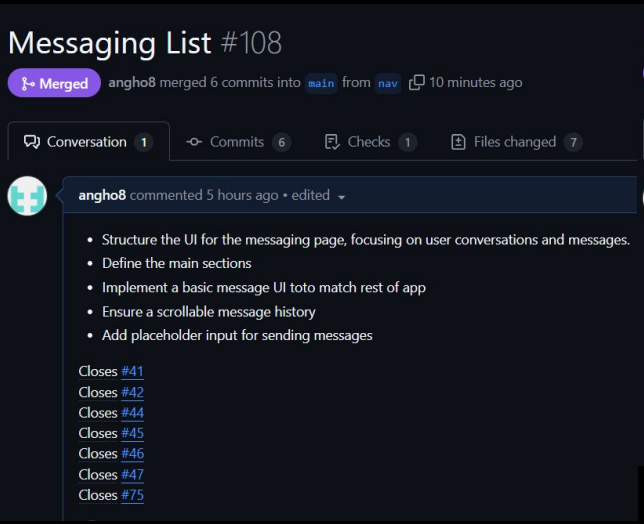
Full Name: Angela Ho
UMass email: angelaho@umass.edu
GitHub ID: angho8
Assigned Work Summary

Login/Sign Up Layout
Create Layout for Login/Sign Up Page
Issues #73, 74, 75, 76, 77, 78
Pull Request #107

Messaging List Layout
Create Layout of Messaging Page
Issues #41, 42, 44, 45, 46, 47
Pull Request #108

Filter Listing Layout
Add filters and sorting UI
Issue #91
Pull Request #95





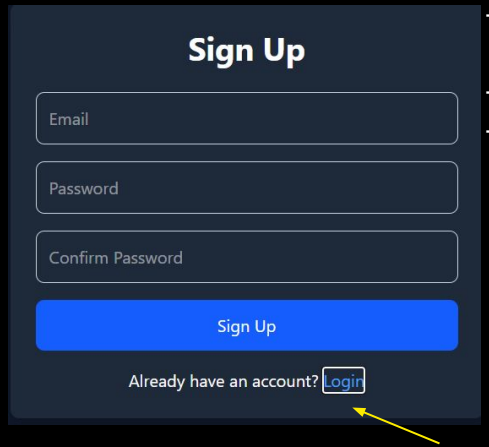
Angela Ho (2/3)

Full Name: Angela Ho
UMass email: angelaho@umass.edu
GitHub ID: angho8
Code & UI Explanation

Login and Sign Up Page

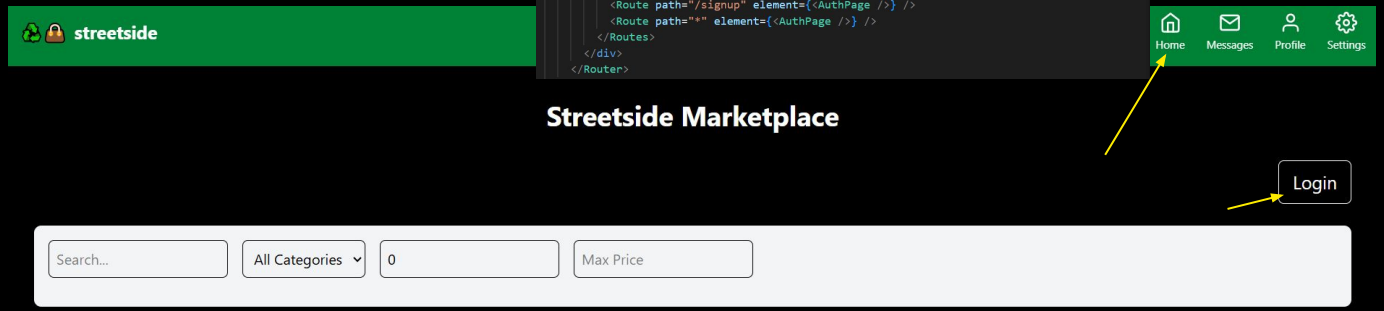
Ensured functionally between the Login and Signup Page

- Navigate from signup and login page easily
- When login successful, automatically brings you to the home page
- Login anytime through the corner login button
- Easily create an account using you email

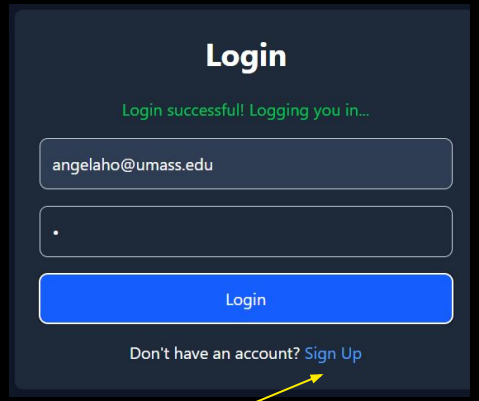



```tsx
import { useNavigate } from 'react-router'
import React, { useState } from 'react'

const AuthPage: React.FC = () => {
  const [isLogin, setIsLogin] = useState(true)
  const [formData, setFormData] = useState({
    email: '',
    password: '',
    confirmPassword: '',
  })
  const [error, setError] = useState<string | null>(null)
  const [success, setSuccess] = useState<string | null>(null)

  const navigate = useNavigate()

  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setFormData({ ...formData, [e.target.name]: e.target.value })
    setError(null)
    setSuccess(null)
  }

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault()
    setError(null)
    setSuccess(null)
```

```tsx
import React from 'react'
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router'
import AuthPage from './Auth'

const AuthNavigator: React.FC = () => {
  return (
    <Router>
      <div className="w-full min-h-screen flex flex-col items-center justify-center bg-gray-9
        <h1 className="text-4xl font-bold mb-6">
        </h1>
        <nav className="mb-6">
          <Link to="/login" className="text-blue-400 hover:underline mx-2">
            Login
          </Link>
          <Link to="/signup" className="text-blue-400 hover:underline mx-2">
            Sign Up
          </Link>
        </nav>
        <Routes>
          <Route path="/login" element={<AuthPage />} />
          <Route path="/signup" element={<AuthPage />} />
          <Route path="*" element={<AuthPage />} />
        </Routes>
      </div>
    </Router>
```



Streetside Marketplace



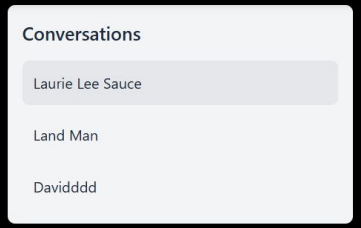
Angela Ho (2/3)

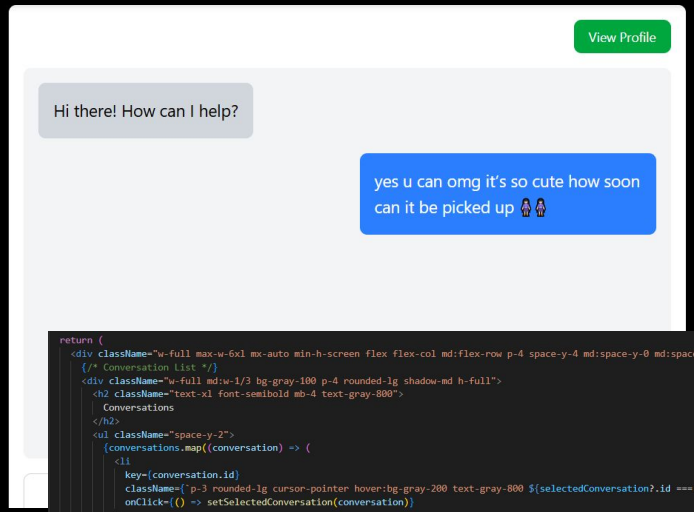
Full Name: Angela Ho
UMass email: angelaho@umass.edu
GitHub ID: angho8
Code & UI Explanation

Messaging List

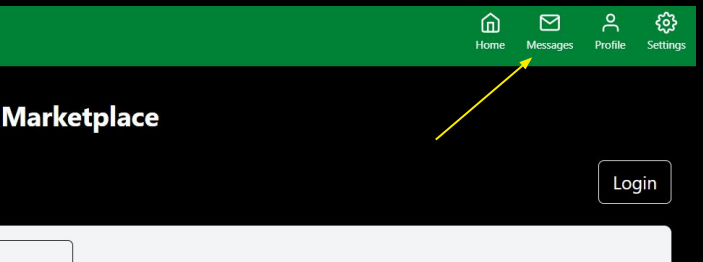
Created UI of Seller and Buyer List of Messages

- Navigate from home page to messages
- Ensure a history of all those contacted on the left side list
- Easily see any chat history and continuing chat highlighting name when selected
- Making sure layout style matches with direct messaging (from Laurie)





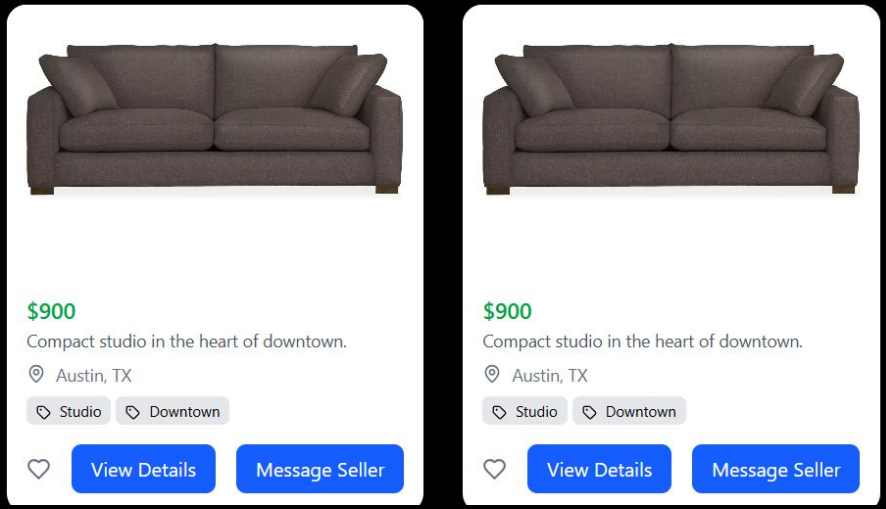

```
return (
  <div className="w-full max-w-6xl mx-auto min-h-screen flex flex-col md:flex-row p-4 space-y-4 md:space-y-0 md:space-x-4">
    {/* Conversation List */}
    <div className="w-full md:w-1/3 bg-gray-100 p-4 rounded-lg shadow-md h-full">
      <h2 className="text-xl font-semibold mb-4 text-gray-800">
        Conversations
      </h2>
      <ul className="space-y-2">
        {conversations.map((conversation) => (
          <li
            key={conversation.id}
            className={`p-3 rounded-lg cursor-pointer hover:bg-gray-200 text-gray-800 ${selectedConversation?.id === conversation.id ? 'bg-gray-300' : ''}`}
            onClick={() => setSelectedConversation(conversation)}
          >
            {conversation.name}
          </li>
        ))}
      </ul>
    </div>

    {/* Message View */}
    <div className="flex-grow bg-white p-4 rounded-lg shadow-md flex flex-col">
      {selectedConversation ? (
        <>
          <div className="flex justify-between items-center mb-4">
            <h2 className="text-2xl font-bold">
              Chat with {selectedConversation.name}
            </h2>
            <Link to="/profile">
              <button className="px-4 py-2 bg-green-600 text-white rounded-lg hover:bg-green-700 text-sm">
                View Profile
              </button>
            </Link>
          </div>

          <div className="flex flex-col flex-grow space-y-4 bg-gray-100 p-4 rounded-lg overflow-y-auto max-h-[60vh]">
            {selectedConversation.messages.map((msg, index) => (
              <MessageBubble
```



Angela Ho (2/3)

Full Name: Angela Ho
UMass email: angelaho@umass.edu
GitHub ID: angho8
Code & UI Explanation

Filtering UI



Created UI for Filtering Listings

- Have multiple options
- Allow to search by title, category selection, price range, and condition
- Different options for prices,, newest/oldest, and distance.
- Filters and sorting are applied in real-time as users make changes.

```
// Filtering
if (filters.searchTerm) {
  updatedListings = updatedListings.filter((listing) =>
    listing.title.toLowerCase().includes(filters.searchTerm.toLowerCase())
  )
}
if (filters.category) {
  updatedListings = updatedListings.filter((listing) =>
    listing.tags.includes(filters.category)
  )
}
if (filters.minPrice || filters.maxPrice !== 9999999) {
  updatedListings = updatedListings.filter(
    (listing) =>
      listing.price >= filters.minPrice && listing.price <= filters.maxPrice
  )
}
if (filters.condition) {
  updatedListings = updatedListings.filter(
    (listing) => listing.condition === filters.condition
  )
}

// Sorting logic
switch (sortOrder) {
  case 'price_asc':
    updatedListings.sort((a, b) => a.price - b.price)
    break
  case 'price_desc':
    updatedListings.sort((a, b) => b.price - a.price)
    break
  case 'newest':
    updatedListings.sort(
      (a, b) => new Date(b.date).getTime() - new Date(a.date).getTime()
    )
    break
```

```
return (
  <div className="container mx-auto p-4 bg-lime-500 text-black min-h-screen flex items-center justify-center">
    <div className="bg-lime-400 shadow-lg rounded-lg p-6 max-w-md mx-auto">
      <h3 className="text-lg font-semibold mb-4">Filters</h3>

      <input
        type="text"
        placeholder="Search..."
        value={filters.searchTerm}
        onChange={(e) =>
          setFilters({ ...filters, searchTerm: e.target.value })
        }
        className="w-full p-2 mb-3 border border-gray-300 rounded-lg bg-white text-black"
      />

      <select
        value={filters.category}
        onChange={(e) => setFilters({ ...filters, category: e.target.value })}
        className="w-full p-2 mb-3 border border-gray-300 rounded-lg bg-white text-black"
      >
        <option value="">All Categories</option>
        <option value="Furniture">Furniture</option>
        <option value="Electronics">Electronics</option>
        <option value="Clothes">Clothes</option>
        <option value="Shoes">Shoes</option>
        <option value="Home Goods">Home Goods</option>
      </select>

      <input
        type="number"
        placeholder="Min Price"
        value={filters.minPrice || ''}
        onChange={(e) =>
          setFilters({ ...filters, minPrice: Number(e.target.value) })
        }
        className="w-full p-2 mb-3 border border-gray-300 rounded-lg bg-white text-black"
```

Angela Ho (2/3)

Full Name: Angela Ho
UMass email: angelaho@umass.edu
GitHub ID: angho8
Code & UI Explanation

The previous slides showcased the components I worked on.

Interaction with Laurie and Landy
After a user signs up or logs in, they're redirected to the grid listing home page (created in collaboration with Laurie and Landy). On this page, they can filter listings by search term, category, price range, and condition, and sort them by price, date, or distance. This filtering and sorting functionality was ensured by Laurie. From there, users can navigate to the messages page via the navigation bar (which Landy implemented). Additionally, the login button is easily accessible from the homepage at all times to ensure smooth user transitions.

Interaction with Laurie's Messaging
For messaging, I designed the layout that allows users to view a list of ongoing conversations on the left and chat history on the right, allowing you to continue conversations easily.When a conversation is selected, it highlights the user and displays the message thread. I focused on making sure the styling remained consistent with the direct messaging view on the item displayed, based on Laurie's design direction.

Interaction with David
This listings, messages, and users rely on the mock data generated by David. This mock data is retrieved from an API built on IndexedDB, providing the necessary data for listings and messaging.

Angela Ho (3/3)

Full Name: Angela Ho
UMass email: angelaho@umass.edu
GitHub ID: angho8
Challenges and Insights

Obstacles + Lessons

- There was a lot of initial struggle with understanding React's state management and methods but as the project progressed I learned a better understanding of hooks like useState and useEffect.
- I realized the importance of component reliability and how to break down UI element into a bunch of pieces that were more manageable to do rather than trying to do all in one.
- It took a bit to master Github, there were a lot more components on it then I have previously used it for especially for a group project. It took time managing pull requests, resolving merge conflicts, and understanding proper branch strategies.

Takeaways

- I gained valuable experience working with a team that came from different backgrounds, coordinating, and managing dependencies across different parts of the project while making sure it all integrates in one.
- Communication was key, making sure that every individual component integrated and especially adding comments on the pull requests to have understanding of what the other team members ave done with their part.
- We really had to pay attention dn adapt to other team members designs and code ensuring alignment on both functionality and UI consistency.

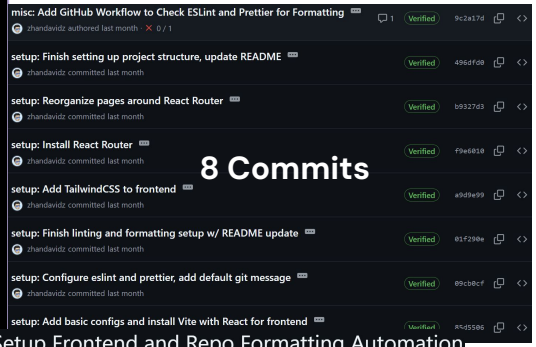
David Zhan (1/3)

Full Name: David Zhan
UMass email: dzhan@umass.edu
GitHub ID: zhandavidz

Assigned Work Summary

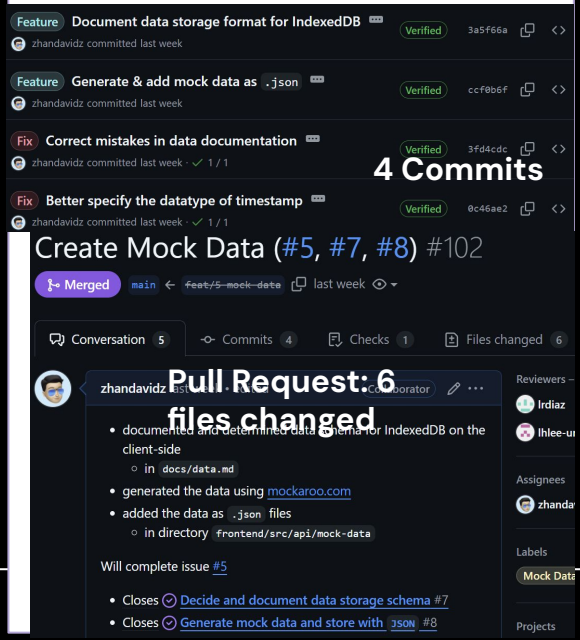
Setup Frontend with Tooling

Setup Frontend Main Issue (#83), which also completes subissues #84, #86
Associated Pull Request



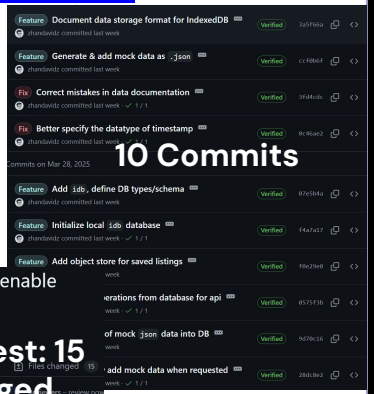
Create Mock Data

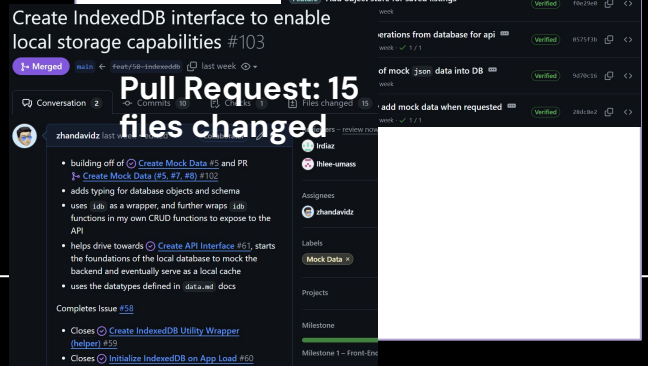
Create Mock Data Main Issue (#5), which also completes subissues #7, #8
Associated Pull Request



IndexedDB Interface

Create IndexedDB Mock Data Interface Main Issue (#58), which also completes subissues #59, #60
Associated Pull Request





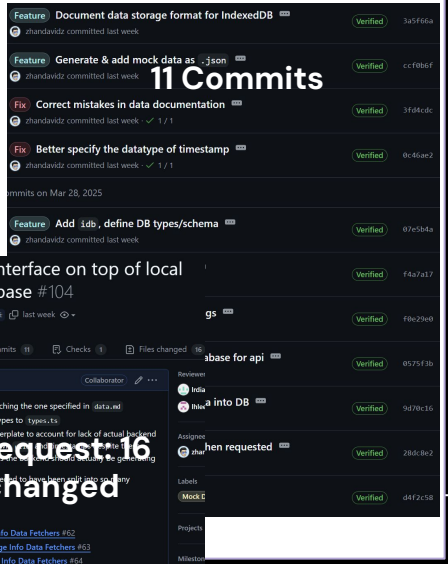
David Zhan (1/3)

Full Name: David Zhan
UMass email: dzhan@umass.edu
GitHub ID: zhandavidz

Assigned Work Summary

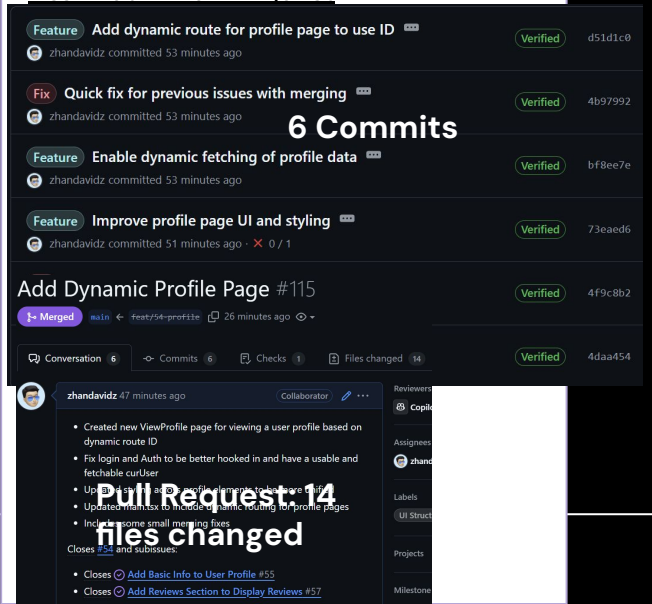
Create API Interface for rest of code to fetch data from

Create API Interface Main Issue (#61),
which also completes subissues #62, #63, #
#65
Associated Pull Request



Update the Profile Page Layout + Styling

Create Layout of User Profile Main Issue (#54),
which also completes subissues #55, #57
Associated Pull Request

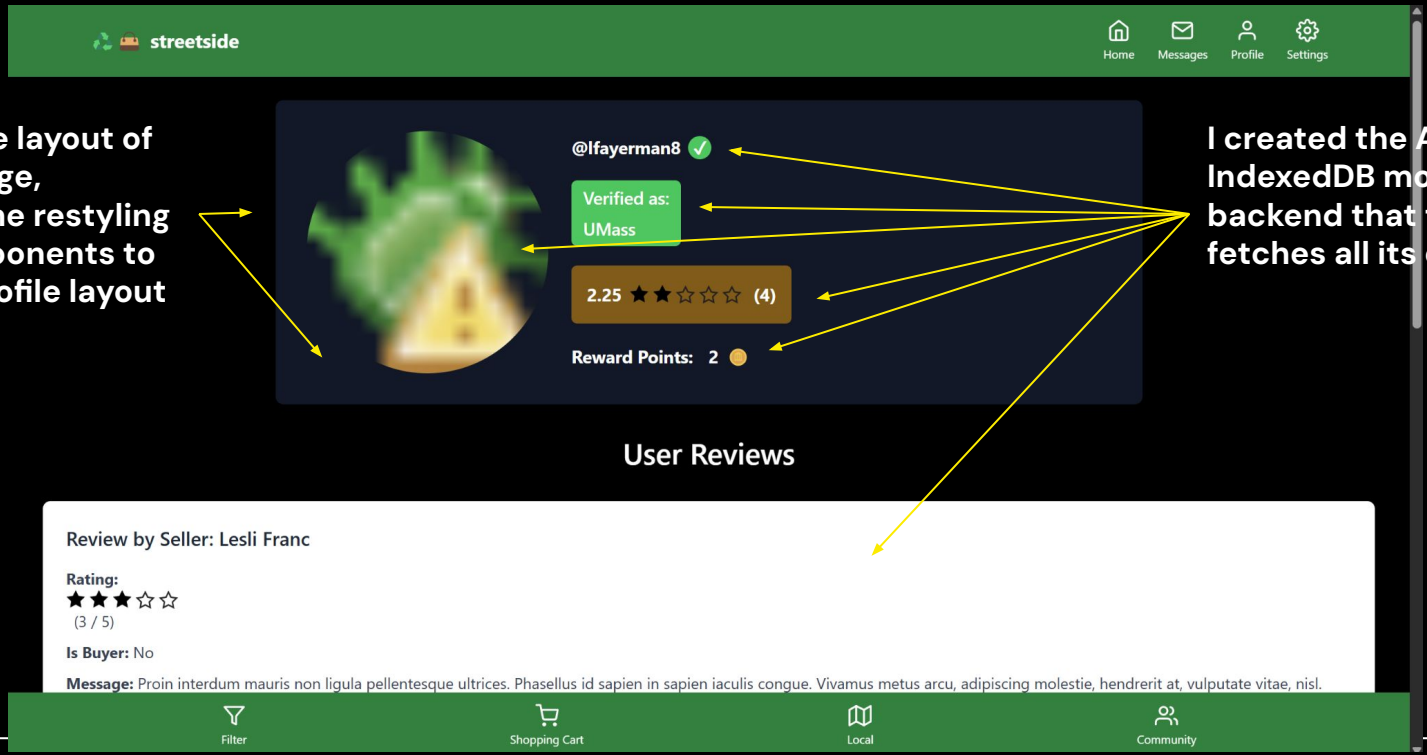


David Zhan (2/3)

Full Name: David Zhan
UMass email: dzhan@umass.edu
GitHub ID: zhandavidz

Screenshots & Demonstration

I designed the layout of the profile page, including some restyling of a few components to fit the new profile layout



I created the API & IndexedDB mock backend that this page fetches all its data from

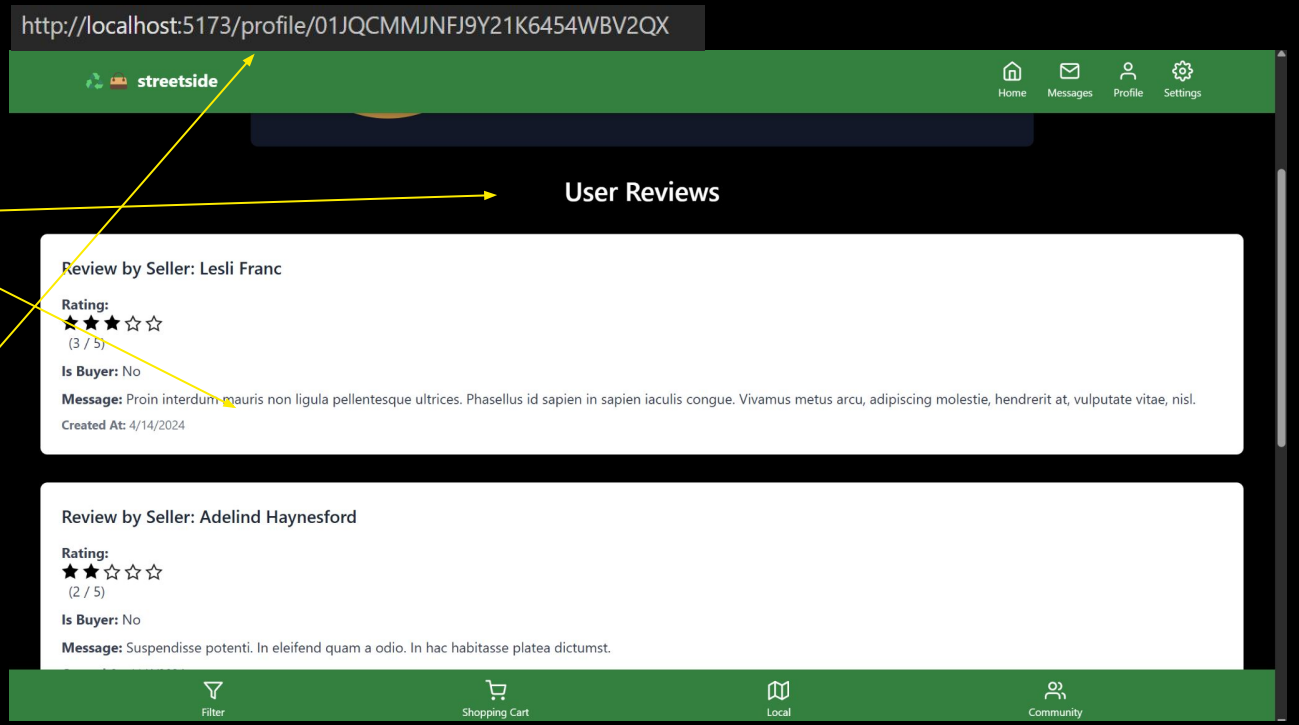
David Zhan (2/3)

Full Name: David Zhan
UMass email: dzhan@umass.edu
GitHub ID: zhandavidz

Screenshots & Demonstration

I also restyled the reviews section to scroll with the page, making it feel less crowded

Additionally, I made the profile page route dynamic so that the /profile page can display the profile of any user given their ID in the url

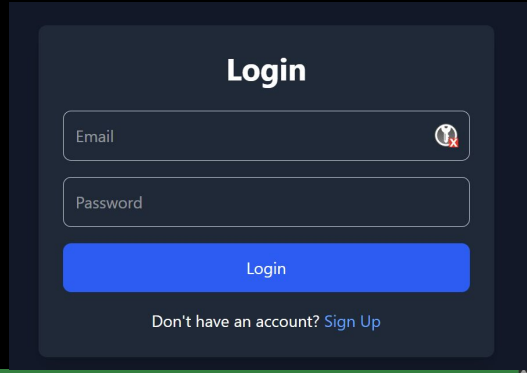


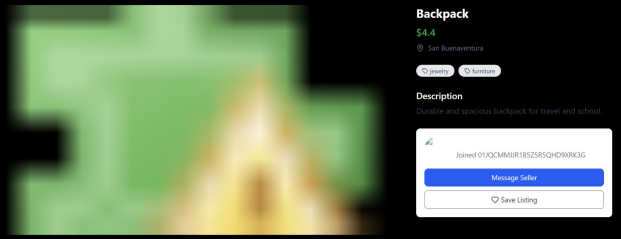
David Zhan (2/3)

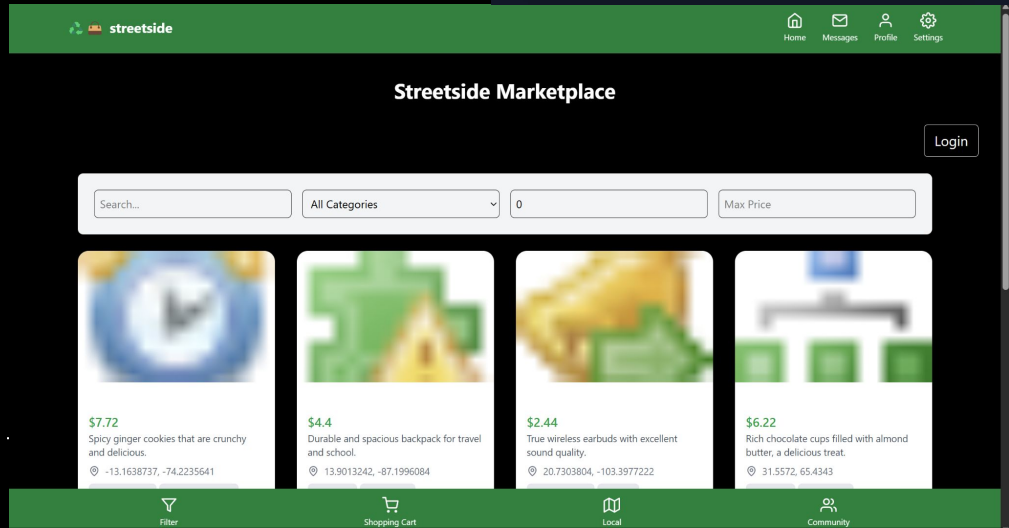
Full Name: David Zhan
UMass email: dzhan@umass.edu
GitHub ID: zhandavidz

I did not design the UI components for the marketplace, individual listing, or the login, but I did design the API interface that they all use to fetch mock data. I also designed the IndexedDB interface that the API uses, as well as created the actual mock data itself. My code helped all of this UI generate elements dynamically and be ready for a real backend.

Screenshots & Demonstration







David Zhan (2/3)

Full Name: David Zhan
UMass email: dzhan@umass.edu
GitHub ID: zhandavidz

Code & UI Explanation

```
async function initDB(): Promise<IDBPDatabase<LocalDB>> {
  try {
    console.log('Initializing database...')
    const database = await openDB<LocalDB>(DB_NAME, DB_VERSION, {
      upgrade(db) {
        db.createObjectStore('users', { keyPath: 'user_id' })
        db.createObjectStore('messages', { keyPath: 'message_id' })
        db.createObjectStore('reviews', { keyPath: 'review_id' })
        db.createObjectStore('listings', { keyPath: 'listing_id' })
        db.createObjectStore('savedListings', { keyPath: 'listing_id' })
      },
    })
    console.log('Database initialized successfully')

    console.log(env.VITE_USE_MOCK_DATA)

    if (!isMocking) {
      isMocking = true
      const userCount = await database.count('users')

      // want to use mock data and have no mock data
      if (env.VITE_USE_MOCK_DATA && userCount === 0) {
        console.log('Initializing database with Mock Data...')
        initializeMockData(database)
        console.log('Database initialized with Mock Data!')
      }
    }

    return database
  } catch (err) {
    console.error('Error initializing database:', err)
    throw new Error('Failed to initialize database')
  }
}
```

This code initializes the database with object stores for each "table" and optionally triggers an insert of mock data into the table if specified by an environment variable (the environment variable is true if the website is started with `npm run dev:mock`).

This is a critical part of the entire application, since every part of the UI hooks into the mock backend and depends on the functionality of the IndexedDB database to store and return proper data. Additionally, since most of the UI would look empty and incomplete without any users or listings, adding mock data into the tables helps make the website feel more fleshed out and enables testing and showcasing of the progress we have made with the site even though there aren't any real users yet.

I faced many challenges making the mock data optional, as I wanted the app to be able to be run without mock data if we didn't want it. Through lots of time and research, I found that I could set a Vite environment variable through the command line, enabling me to make separate `npm` scripts that use different environment variables without having to manually change it in a `.env` file. This ~~is not a UI component, so there were no styling guidlines other than adhering~~ to documentation that I created for the interface.

David Zhan (2/3)

Full Name: David Zhan
UMass email: dzhan@umass.edu
GitHub ID: zhandavidz

Code & UI Explanation

This code exposes the listings in the IndexedDB database to all other components through an API. It is type-safe and interfaces correctly with IndexedDB to carry out many commonly used functions that other components may want.

It enables the marketplace and single listing view to function, as it provides them an API endpoint to fetch data from. While I didn't write the UI components relevant to listings, I communicated with the team members writing that code to ensure that relevant endpoints were exposed.

I faced many challenges figuring out how to properly interface with the IndexedDB interface, as well as how to manage uuids (which I found out have now been replaced by ulids) as well as storing the current time in epoch. I fixed these through research, importing the ulid library to help me with generating IDs and making a helper function that abstracts away the process of getting the current time in Epochs.

This is also not a UI component, so there is no styling to adhere to.

```
export const Listings: ListingsAPI = {
  async getListing(listingId) {
    return listingsDB.get(listingId)
  },
  async getListings() {
    return listingsDB.getAll()
  },
  async getListingsFromUser(userId) {
    const all = await listingsDB.getAll()
    return all.filter((l) => l.seller_id === userId)
  },
  async createListing(listing) {
    // FIXME: when there is a backend, make sure backe
    backend, then store that in local db with ulid fro
    const newListing = {
      ...listing,
      listing_id: ulid(),
      created_at: curEpoch(),
    }
    await listingsDB.create(newListing)
    return newListing
  },
  async updateListing(listing) {
    const newListing = {
      ...listing,
      updated_at: curEpoch(),
    }
    await listingsDB.update(newListing)
    return newListing
  },
  async deleteListing(listingId) {
    return listingsDB.delete(listingId)
  },
  async saveListing(listingId) {
    await savedListingsDB.create({
      listing_id: listingId,
      user_id: curUser!.user_id,
      saved_at: curEpoch(),
    })
  },
  async unsaveListing(listingId) {
    await savedListingsDB.delete(listingId)
  },
}
```

David Zhan (2/3)

Full Name: David Zhan
UMass email: dzhan@umass.edu
GitHub ID: zhandavidz

Component Hierarchy & Interaction

My coding contributions were mostly centered around developing the framework and basis on which UI components could be successfully developed. I focused on making a sustainable workflow with checks in the GitHub repository. Then, I spent a lot of time working on mock data, generating it with Mockaroo and then serving it with IndexedDB and an API interface layer.

On start up, the `initDB` function gets called, which creates the IndexedDB object stores. It checks for an environment variable for mock data, and fills IndexedDB with mock data if the environment variable is set to true. With this, the mock data is set up.

Now, any component can import the API and fetch data by just calling functions in the API. The API interfaces directly with IndexedDB through my custom IndexedDB wrapper, applies some post-processing, and completes the CRUD action asked of it. In this way, components only need to call an API that abstracts away the data fetching, and the API can handle getting the data or updating the data. In the future, there will be an extra step here where the API will also abstract away the processes of fetching from the backend and synching between the frontend IndexedDB offline store and the backend microservices. All of these updates can be done without any changes to the frontend components, since this data-fetching process is abstracted away from the components.

David Zhan (3/3)

Full Name: David Zhan
UMass email: dzhan@umass.edu
GitHub ID: zhandavidz

Challenges and Insights

I faced many challenges with creating the mock data interface. I didn't realize initially how time-consuming this process would be, especially with the generation of the data. Originally, I had even wanted to generate actual mock images for all the people and listings, but I quickly found out that that wasn't worth the time investment when I don't need real images to demonstrate frontend component functionality. Additionally, I found out quickly that dealing with IndexedDB directly was a pain, and that using a wrapper such as `idb` made my life a lot easier. Finally, I found that predicting possible API endpoints that may be needed was very difficult.

One big thing that I have learned is that the process of creating API endpoints requires a lot of synchronous collaboration between the API creator and the team members using the API, and that it is often more efficient to just create API endpoints as we go, since we are constantly realizing new data endpoints that could be useful mid-development, and many endpoints created originally do not end up being used.

Overall, I found that working in a team environment helped make things go faster since there are more people working on things at the same time, but also that a lot of the time saved gets lost when trying to sync between team members with busy schedules, and merging code and ideas sometimes can be difficult. Overall, I really enjoyed the experience, but there are always places to improve upon in collaboration in the future!

David Zhan (3/3)

Full Name: David Zhan
UMass email: dzhan@umass.edu
GitHub ID: zhandavidz

Future Improvements & Next Steps

- I believe there are some issues around data throughout the app, and I am not 100% sure if all features are using the mock data interface yet.
- I believe auth has been done enough to be functional, but I believe there are definitely improvements that can be made, which will probably be made easier when we have an authentication microservice.
- I think the API endpoints can probably be better fine-tuned to the specific needs of each feature, since there are some endpoints I thought would be needed that didn't end up being needed.
- Though we covered all the issues, I think there should be more features involved with creating, updating, and deleting things. Specifically, I think it could be nice to be able to delete and update some of your own listings, update your profile information, and update and delete reviews, amongst other things.
- One thing I just realized was that, despite having a place to show reviews, we forgot to implement or even make an issue for creating reviews, so we should probably do that.

Future Improvements & Next Steps

Laurie Lee Thoughts

I think that improvements could be made on generating more transitions and features for out application. It would be cool to add a progress bar as a user navigates between pages. For example, when a user wants to message a seller and they click “Message Seller”, there could be a progress bar indicating how close they are to reaching the DM page. Some areas of technical debt that could be optimized further include more error handling (e.g., “failed to load data” or “null reference error” instead of getting a terminal error).

Angela Ho Thoughts

I think that improvements to made are adding features that make it easier for users to search for items, like distance filter. Since we want to mainly cater to UMass students, it would help them find items nearby and encourage safety. We can also include a feature like a sticker to include UMass student on their profile. We can enhance the profile section for safety purposes, like allowing users to see interests or verified information.

Landrick Diaz Thoughts

I think that a future improvement that our group can do is not necessarily adding more features, but taking a step back and looking at overall structure and organization of the project. As we progress into the creation of this app and adding more files and functionality, it can become really easy to feel overwhelmed or confused by the growing complexity of the code. Personally, I find it harder to understand and navigate the codebase as it expands, so improving project structure and clarity would help the whole team stay aligned, work more efficiently, and be able to improve functionalities with what we learn in class.

David Zhan Thoughts