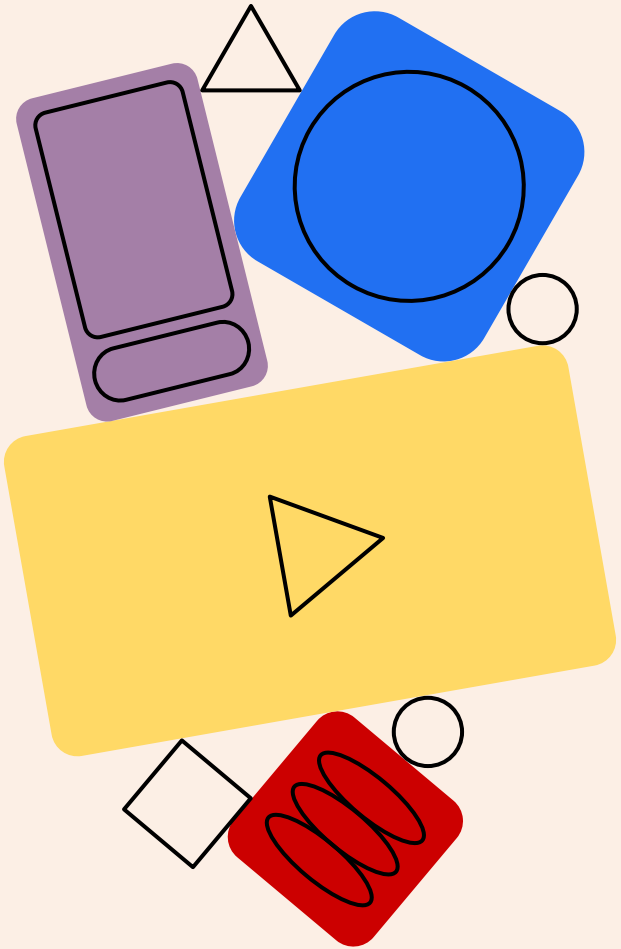
TRAVY

Trips Without the Hassle

# Milestone 1



Date: 04/02/2025

Team Minute Travelers: Tanush Savadi, Ayush Ravi Chandran, Ipsita, Ian McKenna

# Project Overview & Team Members

**Project Overview:**
Travy is a travel planning web application designed to help university students, especially international students, easily organize affordable trips home or to popular destinations during school breaks. The app simplifies the trip-planning process by aggregating multiple transport options, collecting personalized travel preferences, and offering a built-in rideshare community.

**Key Functionalities:**
Travy helps students plan trips easily by offering an interactive landing page with a map and upcoming routes to popular airports, filtered by budget, transport mode (bus, train, rideshare), and travel time. Users can also connect through a location-based community forum to arrange rideshares, post travel plans with preferences and tags, and comment on others' posts. Each rideshare post includes travel details, and users can expand posts to view full descriptions and reply directly.

**Github**: https://github.com/tanushsavadi/travy

**Issues & Milestone**: https://github.com/tanushsavadi/travy/milestone/1

**Team Builders:**

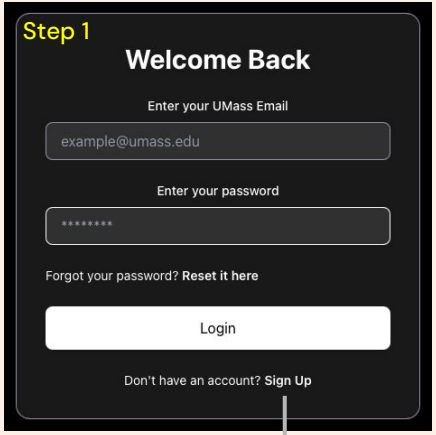
1. **Tanush Vijayakumar Savadi:** Frontend development of Login page, registration page, and user profile creation using Tailwind styling.

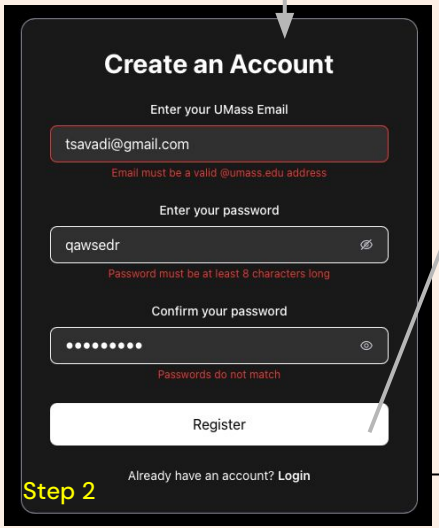
2. **Ayush Ravi Chandran:** Worked on the UI and logic for the main Landing page, which includes an interactive map, filter options and showing transportation options.

3. **Ipsita & Ian McKenna:** Full-stack development on Community section with reusable components.

# Software Architecture Overview

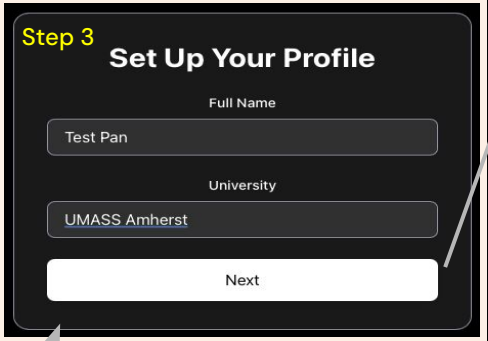
Start: Profile Setup




Step 1: Login Page (/login)

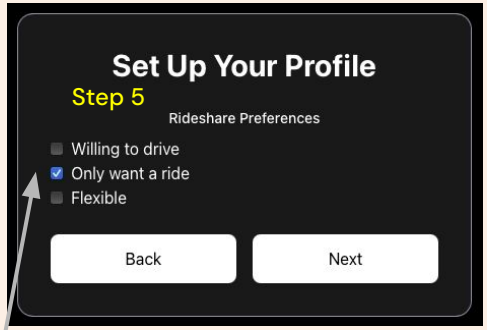
- The user is first presented with a login screen where they enter their UMass email and password.
- If the user already has credentials saved in localStorage, they can be automatically redirected to the dashboard.
- A link is provided to go to the registration page if the user is new.




Step 2: Register Page (/register)

- New users register with a valid UMass email and a password (with validation).
- After successful registration, their credentials are temporarily saved in localStorage.
- They are then redirected to begin the profile setup.


Step 3: Profile Setup – Basic Info (/profile-setup): The user provides full name and university.




End of profile setup process

Step 4: Profile Setup – Preferences

- Users choose their preferred transport modes (using checkboxes).
- They input their budget and add multiple frequent travel destinations dynamically.

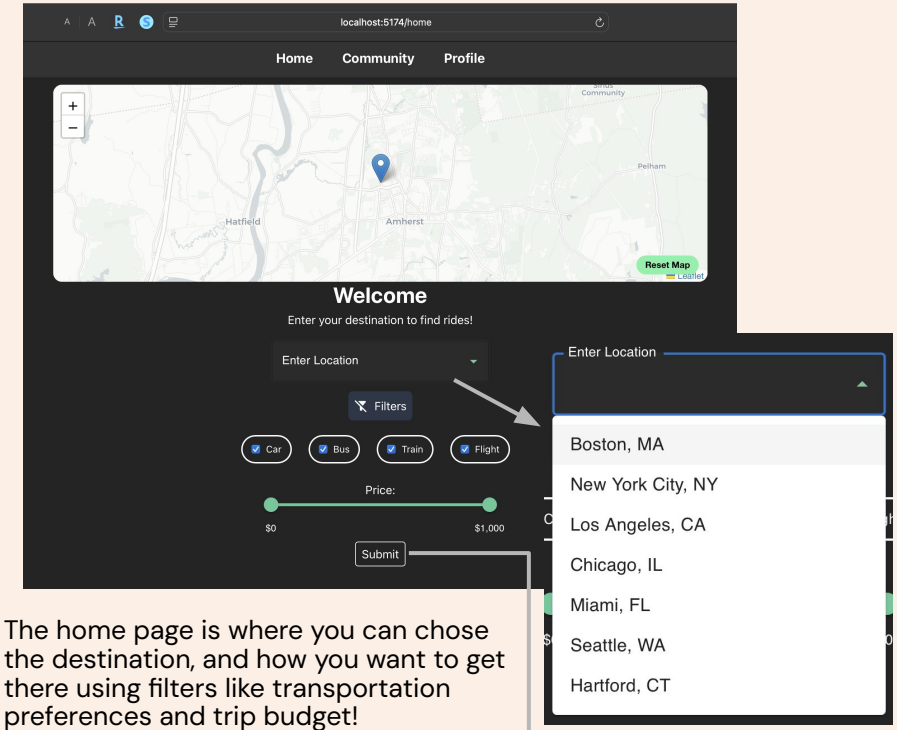
Step 5: Profile Setup – Rideshare Preferences

- Final profile preferences are captured here with rideshare options like:
  - Willing to drive
  - Only want a ride
  - Flexible

Once this step is completed, the user is navigated to the home/dashboard and their complete profile is saved to localStorage.

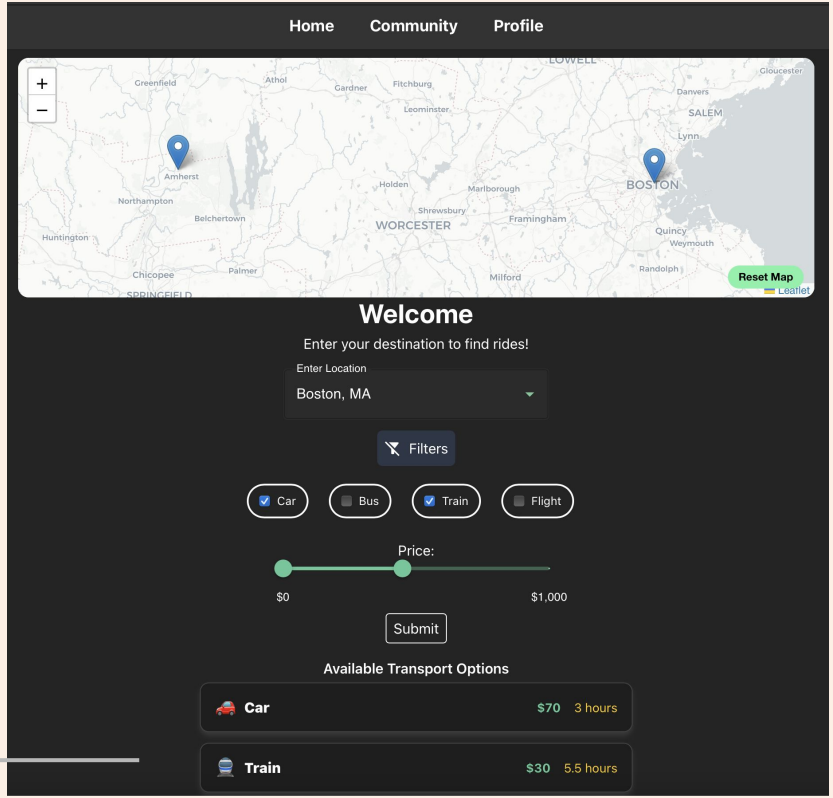
# Software Architecture Overview

After finishing profile creation, you get to this page

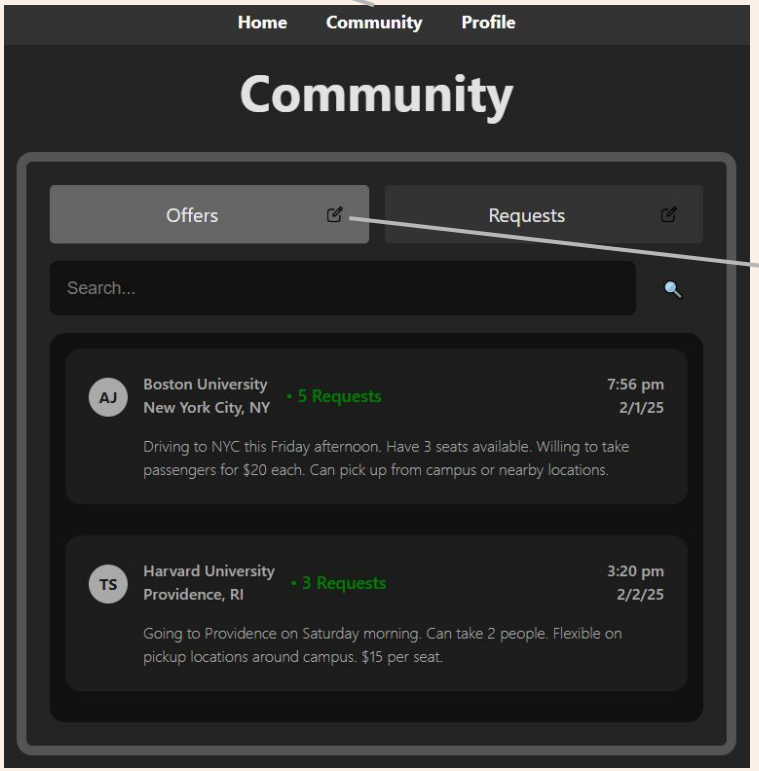



The home page is where you can chose the destination, and how you want to get there using filters like transportation preferences and trip budget!

After choosing your filters, you can hit submit to get a list of available transport options!



**Upon selecting the community tab from the navbar, you get redirected to this page, which shows you posts where people are offering a rideshare service and a requests tab where people are seeking/looking for a rideshare service**





**When the create post icon (the mini pencil on a square box) is selected, you can either create a post where you can chose whether you wish to offer a rude or request one**

# Software Architecture Overview

## Explain how different UI components interact and how state management is handled

**Login → Register → Profile Setup**

- After logging in or registering, users are directed to a multi-step profile setup form(CreateProfile.tsx) which updates the UserProfileContext after every input step.
- Steps include collecting basic info, preferences, destinations, and rideshare intent.
- Credentials stored during registration are synced back using useEffect in CreateProfile to ensure persistence.

**Profile Setup → Home**

- On completion, the user lands on the Home page, where they can use dropdowns, filters, and sliders to specify travel preferences.
- These selections are passed to child components like Map.tsx, TransportOptions.tsx, and Filters.tsx.

**Filters & State Flow**

- The Home component manages the state for filters and destination. It passes these as props to the Map, which dynamically updates pins and transport listings based on the input.
- State is shared using props for responsiveness and accurate visual feedback.

**Community Page**

- Offers and Requests tabs display data in ListedPost components passed from SimpleTabs.
- Users can click the create icon to open the CreatePost modal, which uses internal state and receives props from parent to control visibility and behavior.

**UserProfileContext**: Stores and persists all profile data across pages using useState and localStorage.
**AuthContext**: Tracks whether a user is authenticated and manages login/logout flow using localStorage.
**Component-level State**: Used in form inputs (e.g., useState in LoginPage, CreatePost) for responsiveness and real-time validation.
**Props Drilling**: Controlled from top-level components like Home to children (Map, Filters, etc.) to keep state centralized and easy to debug.

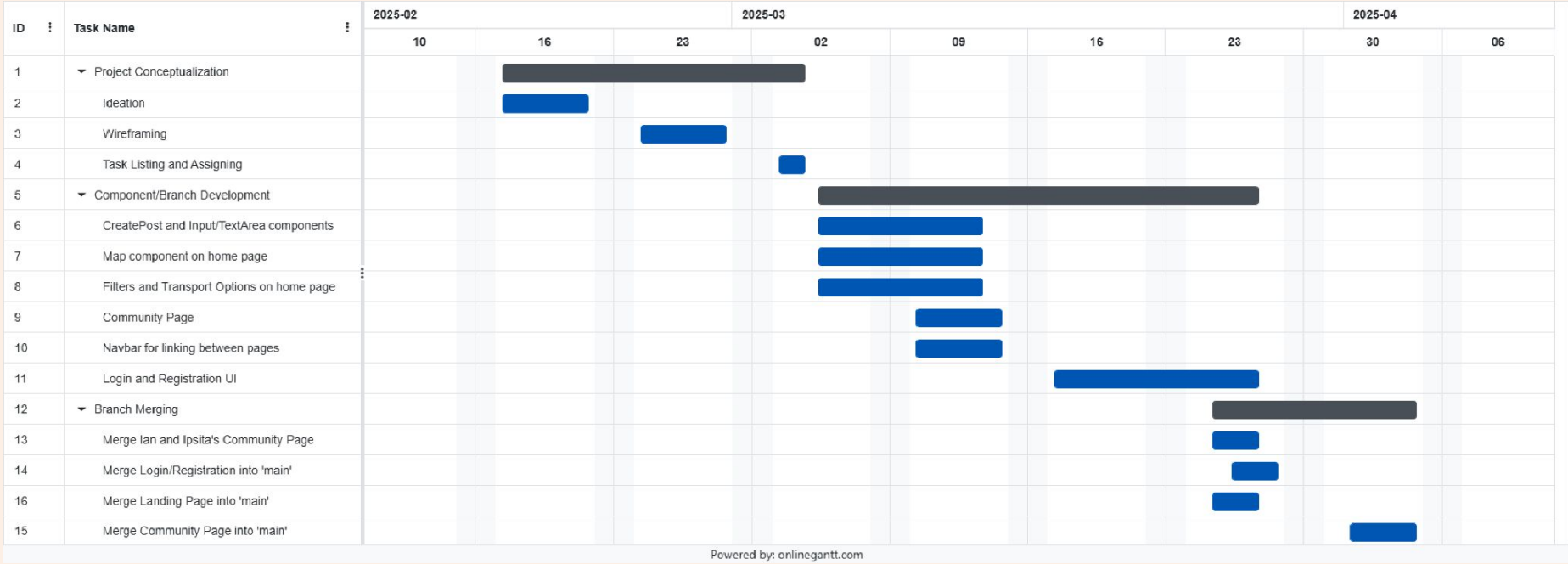
# Historical Development Timeline

**Main Phases:**

Ideation 02/16

Development 03/02

Merge/Release 03/21



# 🎨 Design and Styling Guidelines

## Inspiration

Link: Travy UI/UX Style Guidelines

Material Design- Having consistent layout, spacing and animations.

Apple HIG- Clarity and user-first interface

Microsoft Fluent UI- for accessibility and form standards

## Colors

**Background: #242424** – Deep grey for a modern, low-distraction canvas

**Primary Text: #FFFFFF** - White which provides high contrast for readability

**Accents:** `#7ac59b` - Pulled from our logo for brand consistency

**Gray Shades:** Neutral tones for inputs, dividers, and secondary content

## Frameworks

TailwindCSS for easy to use and consistent styling (consistent spacing and layout `p-4`)
MaterialUI is used to adhere to Google's Material Design

## Typography

**Font**: System UI / sans-serif for native feel and performance. `text-xl` for headings

## Accessibility Considerations

Color contrast meets accessibility standards

Keyboard navigation supported

ARIA labels for icons & semantic alerts for error messages

Added meta tags to enhance usability and SEO.

# Component Documentation

Full Link: Docs

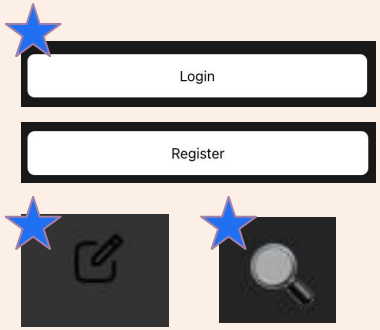
**Button → A reusable button with customizable text and type.**

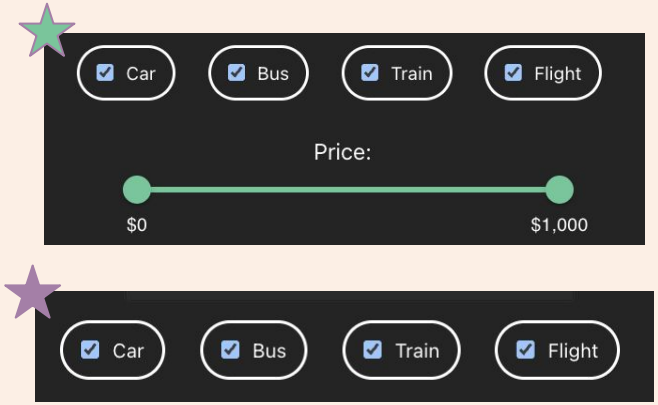
**Checkbox Group → Renders checkboxes for multi-select settings.**

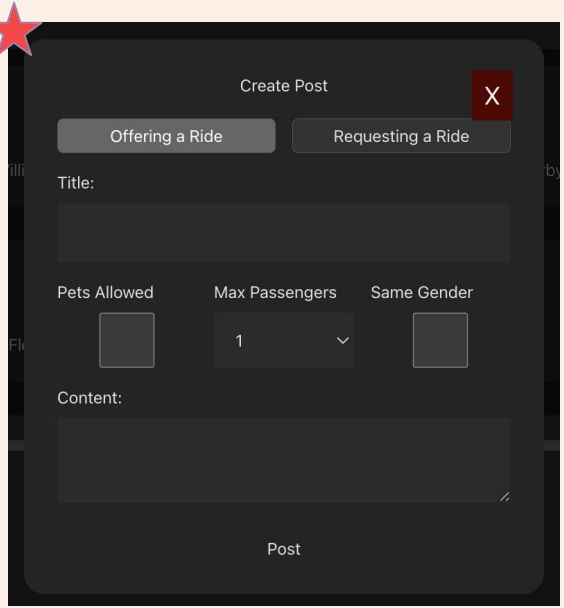
**CreatePost Modal → Form to create a new community post.**

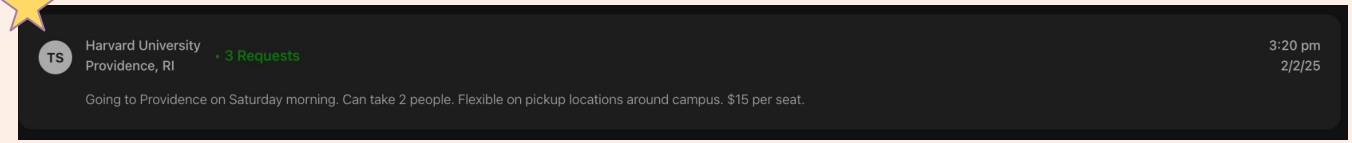
**Filter → Filters a list of items (e.g., trips/posts)**

**ListedPost → Displays a post summary in the community page.**



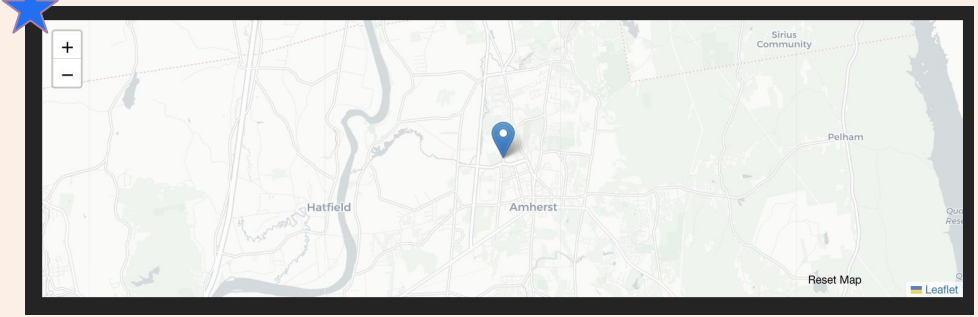
# Component Documentation

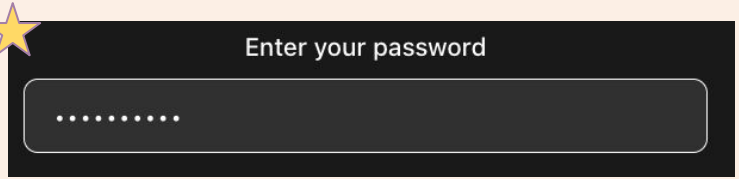
**Map → Displays a static map location.**

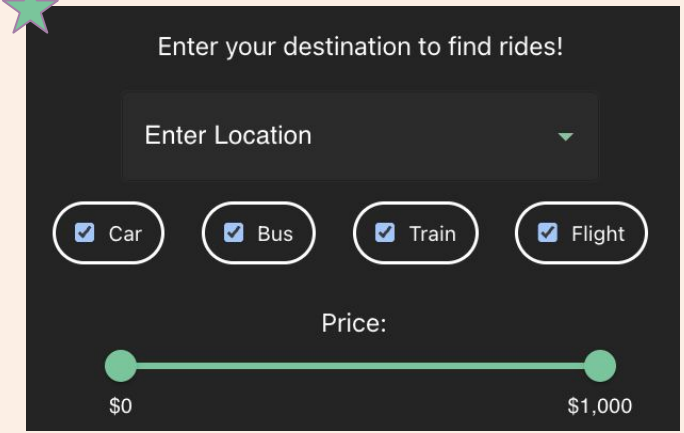
**PasswordInput → Password input with masking**

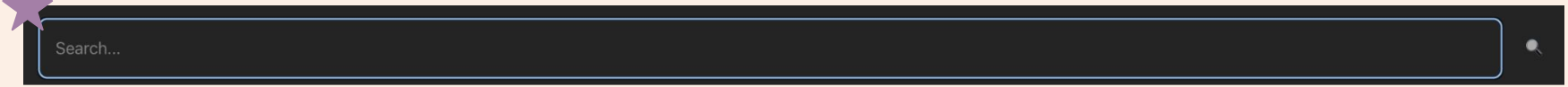
**TransportOptions → Selector for users to be able to filter routes based on transportation modes, price, etc.**

**SearchBar → Used to filter posts that contain keywords/tags that are typed in this search bar**



# Performance Considerations

Conditional Rendering: the CreatePost component is only rendered when needed

```
{showCreatePost && <CreatePost ... />}
```

This prevents unnecessary rendering when the component isn't open/visible.

React Fragments are used whenever a  with no styling is present.

All pages load within 64ms, most at 16ms, according to the Performance tab in Chrome's developer tools.

```
const filteredData: RideOption[] = React.useMemo(() => {
  return rideData.filter(item =>
    item.destination === filters.location &&
    filters.transport.includes(item.transport) &&
    item.price >= filters.price.min &&
    item.price <= filters.price.max &&
    item.travelTime >= filters.travelTime.minHours &&
    item.travelTime <= filters.travelTime.maxHours
  );
}, [filters]);
```

Added lazy loading using `Suspense` and `React.lazy` for all the app pages (as seen in app.tsx) and the TransportOptions component as it is not needed immediately. We only need it when the user has searched for a location. This helps reduce the initial JavaScript bundle size and improves page load speed.

```
<Suspense fallback={<div className="loading">Loading...</div>}>
  <TransportOptions filters={filters} />
</Suspense>
```

We used Tailwind to reduce the amount of CSS files and minimize unused styles. This helps with fast loading of the pages.

Optimized useState and useEffect dependency arrays to only re-render components when necessary

Used the `useMemo()` Hook to cache the results of computations

# Assigned Work Summary - Tanush

🔧 **Closed Issues:**

- **#5** – Implement Login & Registration UI (https://github.com/tanushsavadi/travy/issues/5)
- **#6** – Implement Authentication State Management (https://github.com/tanushsavadi/travy/issues/6)
- **#7** – Implement User Profile Setup UI & Data Collection (https://github.com/tanushsavadi/travy/issues/7)
- **#8** – Ensure Multi-Step Form Navigation for Profile Setup (https://github.com/tanushsavadi/travy/issues/8)

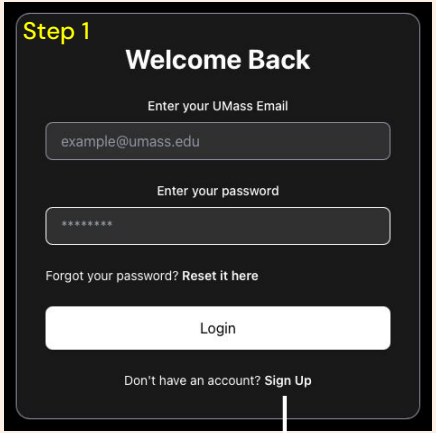
🔨 **Key Contributions:**

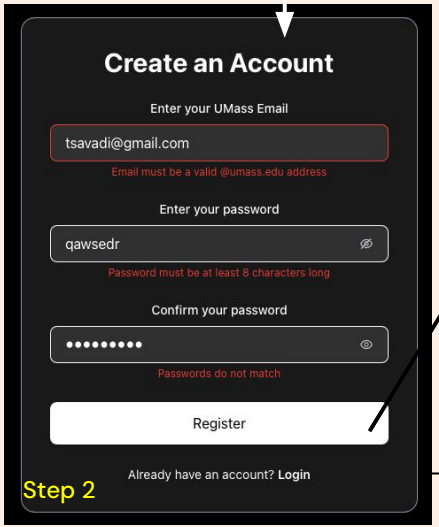
- Built login & registration pages with validation and Tailwind styling
- Developed multi-step profile setup UI with persistent state
- Integrated global authentication with localStorage session tracking
- Verified credentials from both dummy users & registered user data
- Ensured seamless user data flow from registration through profile setup

✅ **Pull Request:**
**#17 – Login and registration**: https://github.com/tanushsavadi/travy/pull/17

# Start: Login Page UI

# Screenshots & Demonstration - Tanush



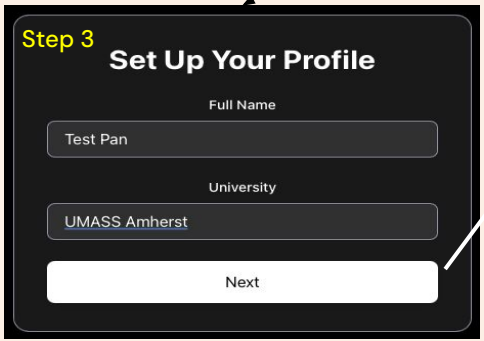




Fields to enter email and password
Reset password option
Sign up, account creation option

Validation checks for email, and password fields

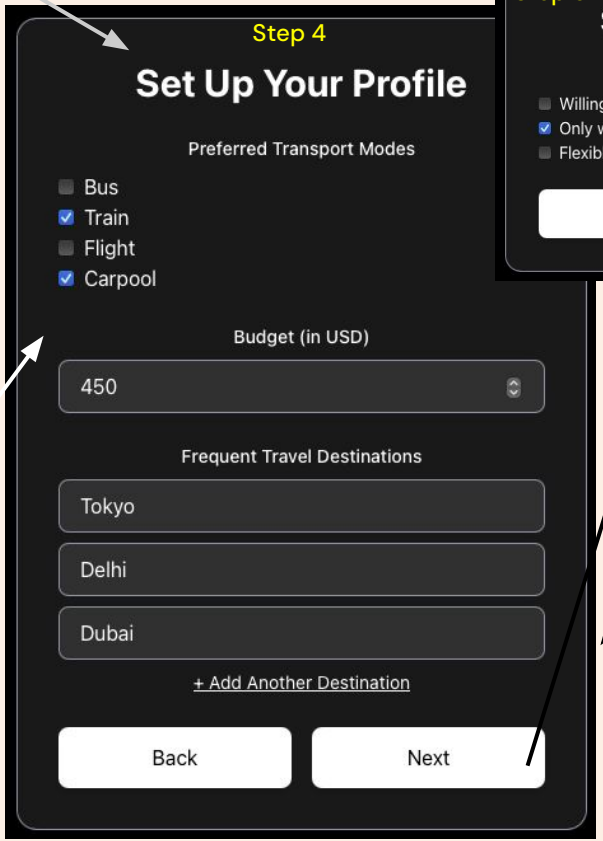
Email: has to have 'umass.edu' to ensure only UMass students can join it

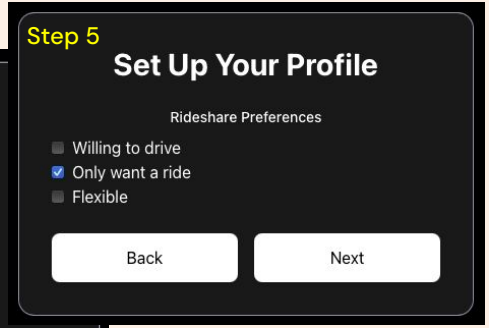
Password: passwords need to be at least 8 characters long and must match on both fields




Enter full name and university fields

Set Up Your Profile – *Main heading guiding the user through profile setup.*




Finish: Profile Setup

**Section 1: Preferred Transport Modes**
Checkboxes:*Allows the user to select multiple travel preferences.*

**Section 2: Budget (in USD)**
Input Field: *Numeric input representing user's budget for transportation.*

**Section 3: Frequent Travel Destinations**
Input Fields: *Captures cities frequently visited by the user.*
Link: + Add Another Destination
*Dynamically adds another destination input field.*

**Navigation Buttons:**
Back – *Navigate to previous step*
Next – *Proceed to the next setup step*

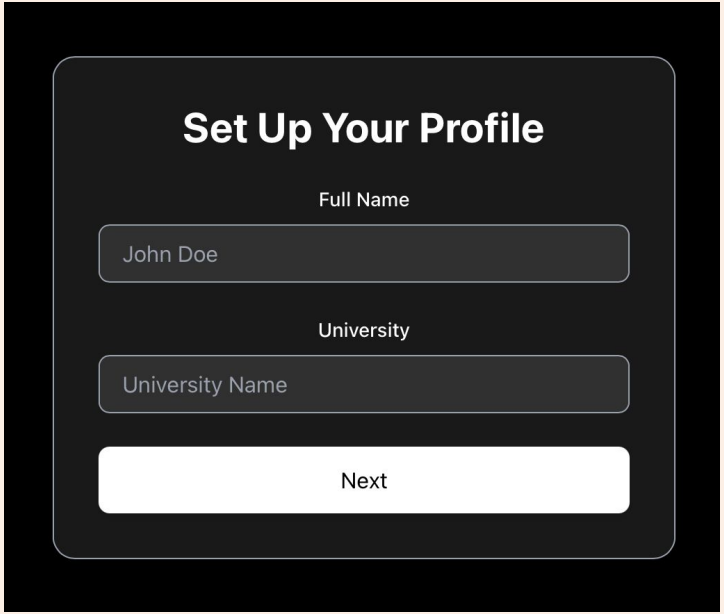
# Code - Tanush

## Code Snippet

src > pages > CreateProfile.tsx > ProfileSetup

```tsx
const ProfileSetup: React.FC = () => {
    <div className="flex min-h-screen items-center justify-center bg-black px-6">
      <div className="w-full max-w-md bg-white/10 backdrop-blur-md rounded-2xl shadow-lg p-8 border
        <h2 className="text-3xl font-bold text-white text-center mb-6">Set Up Your Profile</h2>

        <form className="space-y-6" onSubmit={handleProfileSetup}>
          {/* Step 1: Basic Info */}
          {step === 1 && (
            <>
              <InputField
                label="Full Name"
                type="text"
                value={profile.fullName}
                onChange={(e) => updateProfile({ fullName: e.target.value })}
                placeholder="John Doe"
                error={errors.fullName}
              />
              <InputField
                label="University"
                type="text"
                value={profile.university}
                onChange={(e) => updateProfile({ university: e.target.value })}
                placeholder="University Name"
                error={errors.university}
              />
            </>
          )}

          {/* Step 2: Preferences */}
          {step === 2 && (
            <>
              <CheckboxGroup
                label="Preferred Transport Modes"
```

## UI Impact



# UI Explanation - Tanush

### How This Integrates into the Overall UI Architecture

- The CreateProfile.tsx component is one of the core onboarding flows after a user registers or logs in.
- It pulls and updates user state via the global UserProfileContext, which is accessible by all other components/pages that need profile data (e.g., dashboard, travel suggestions).
- Its role is to initialize and store structured user preferences, which directly influence route filtering, budgeting, and rideshare matchmaking later in the app.
- By centralizing state, the design ensures that subsequent UI components can access accurate and consistent user data without needing to pass props manually.
- A step-based form layout enhances UX by reducing visual clutter and cognitive load, aligning with modern onboarding design principles.
- Styling is consistent with the overall app using Tailwind CSS, with blurred glass effects and contrast-focused color usage (black, white, accent grays).

The CreateProfile.tsx component allows a multi-step user profile setup form, guiding users through entering relevant travel preferences.
The page is split into three distinct steps:

1. **Basic Info** – Full Name & University
2. **Preferences** – Budget, Transport Modes, Destinations
3. **Rideshare Preferences** – Rideshare willingness

State is managed globally via UserProfileContext, allowing data to persist across steps, even on navigation.
Each field is a controlled component, improving validation and input management.

**Challenges & Solutions:** During development, I encountered a few key challenges that I had to address. One issue was that local storage kept resetting between the login and profile setup pages. To solve this, I synced the credentials from `localStorage` into the `UserProfileContext` on component mount using `useEffect`. I also needed a way to maintain user input across multiple form steps, so I used a centralized context to persist state globally. Lastly, to make sure the data was accessible across the app, I ensured every update modified both `useState` and `localStorage` in the context.

# Component Hierarchy & Interaction - Tanush

## 🧩 Component Hierarchy & Interaction

My work fits into the foundational layer of the application's user journey. I implemented:

- LoginPage and RegisterPage — the entry points of the app.
- CreateProfile (multi-step form) — the onboarding flow after registration.

These components rely on:

- UserProfileContext — a global context that stores user information and persists it to localStorage.
- AuthContext — to manage login state and route protection.

All pages use modular components like:

- InputField, CheckboxGroup, PasswordInput, and Button to ensure consistency and reusability across the UI.

## 🔄 User Flow Representation

1. User registers → Credentials saved in localStorage under userProfile.
2. User navigates to profile setup → Form is pre-filled using UserProfileContext which syncs with localStorage.
3. User completes setup → Final profile saved globally and locally.
4. User lands on Home → All data accessible app-wide via context.

This flow ensures a smooth transition across views, while maintaining a persistent and unified user experience.

# Challenges & Insights - Tanush

One of the main challenges I faced was managing form data across multiple steps in the profile setup while ensuring that nothing was lost on navigation. I solved this by leveraging React Context to store user data globally, allowing for smooth, persistent updates across components. Another obstacle was ensuring that credentials from the registration step were accessible in the setup flow. I resolved this by syncing localStorage into UserProfileContext using useEffect.

Working within a collaborative team taught me the importance of clear communication and consistent styling. By modularizing components and maintaining shared state, we ensured our UI stayed cohesive and reusable. This sprint highlighted the value of separating logic from presentation and reinforced how thoughtful state management improves both developer and user experience.

# Future Improvements & Next Steps - Tanush

1. **Secure Password Storage**
   Currently, user passwords are stored in localStorage, which poses a significant security risk. In future sprints, we plan to implement secure authentication using a backend server. This will involve hashing passwords, managing secure sessions or tokens (e.g., JWT), and adhering to best practices in user authentication. This change will align our application with real-world security standards and reduce the risk of data breaches.
2. **Profile Persistence & Multi-User Support**
   At present, only one user profile is stored at a time in localStorage, limiting support for multiple users or concurrent sessions. We aim to persist all user profiles in a database, enabling unique and personalized data to be stored and retrieved per user. This enhancement will allow for smoother onboarding, data continuity, and a scalable foundation as the user base grows.
3. **Backend Integration with Express/Node.js**
   The current implementation is frontend-only, relying on mocked data and local storage. We propose integrating a proper backend using Express and Node.js to manage routes for user authentication (login/register), profile creation, and data retrieval. This would improve performance, data consistency, and maintainability while paving the way for advanced features like search, filtering, and community-driven content.

# Assigned Work Summary - Ayush

🔧 **Closed Issues:**

- **#1** – Create Map Component (https://github.com/tanushsavadi/travy/issues/1)
- **#2** – Create Filter component (https://github.com/tanushsavadi/travy/issues/2)
- **#3** – Create component to display transport options (https://github.com/tanushsavadi/travy/issues/3)
- **#4** – Create Main Landing Page (https://github.com/tanushsavadi/travy/issues/4)

🩹 **Open Issues**:

- **#21** - Add branding to navbar (https://github.com/tanushsavadi/travy/issues/21): I was unable to edit the existing logo on Canva and will need to make a new one from scratch. To be done in the next milestone

🔨 **Key Contributions:**

- Built core UI components for the Landing/Home page including interactive map using Leaflet, filters, and transport listings. Used Material UI and Tailwind for component styling
- Implemented dynamic zooming and adding pins on the map based on user selected location and a reset option
- Integrated easy to use filters to filter transport options (fetched from mock data) based on user's needs in real time
- Added autocomplete data entry box to allow easy selection of location from the mock data

✅ **Pull Request:**

**#16 - Create main Landing Page**: https://github.com/tanushsavadi/travy/pull/16

# Code - Ayush

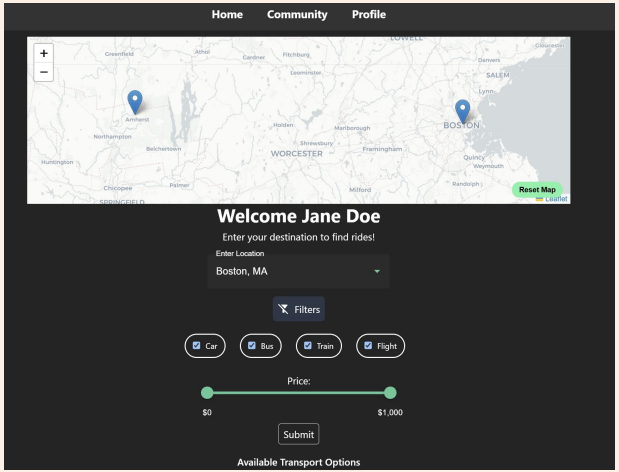
## Code Snippets

src > components > Map.tsx > Map

```tsx
const Map: React.FC<MapProps> = ({ destination, setDestination }) => {
    const ResetButton: React.FC = () => {
        const mapRef = useMap()
        const resetMap = () => {
            if (mapRef) {
                mapRef.setView(UMassCoords, 12); // Reset coordinates
                if (setDestination) setDestination(''); // Reset destination
            }
        };
        const isSmallScreen = window.innerWidth <= 768;
        return (
            <button
                onClick={resetMap}
                className="bg-green-300 hover:bg-green-500 text-white font-bold
                style={{ color: 'black', position: 'absolute', bottom: '10px', right:
            >
                Reset Map
            </button>
        );
    };

    return (
        <div>
            <MapContainer
                center={UMassCoords}
                zoom={12}
                scrollWheelZoom={true}
                className="map-container"
            >
                <TileLayer
                    url="https://{s}.basemaps.cartocdn.com/light_all/{z}/{x}/{y}{r}.pn
                />
                <ResetButton />
                <Marker position={UMassCoords}>
                    <Popup>
                        UMass Amherst
                    </Popup>
                </Marker>
                <DestinationMarker destination={destination} />
```

```tsx
const Home: React.FC = () => {
};

return (
  <div className="text-white">
    <div
      style={{
        position: 'relative',
        overflow: 'hidden',
        width: '60%',
        margin: 'auto',
        ...(window.innerWidth <= 768 && { width: '100%', margin: '0'})
      }}
    >
      <Map destination={destination} setDestination={setDestination} />
    </div>
    <div
      style={{
        overflowY: 'auto',
        width: '80%',
        margin: '0 auto',
        textAlign: 'center',
        ...(window.innerWidth <= 768 && { width: '100%' })
      }}
    >
      {currentUser &&
        <h1 className="text-xl sm:text-3xl font-bold mb-1">Welcome {currentUser.fullName}</h1>
      }
      <p className="mb-4 text-md">Enter your destination to find rides!</p>
      <Autocomplete
        disablePortal
        options={locationNames}
        sx={{
          width: 300,
          margin: '0 auto',
          padding: 0,
          textColor: 'white',
          marginBottom: '0.75rem',
          '& label': { color: 'white' },
          '& label.Mui-focused': { color: 'white' },
          '.MuiAutocomplete-input': { color: 'white' },
```

## UI Impact



# UI Explanation - Ayush

## How This Integrates into the Overall UI Architecture

- The Map.tsx component serves as a visual and interactive centerpiece within the main user-facing home page.
- `Map.tsx` is a modular, reusable component that can be easily dropped into any page layout.
- By consuming `destination` as a prop, often sourced from global or shared state like `UserProfileContext` or page-level state, it stays in sync with user selections made in other components like filters or search bars.
- It uses `mockLocations` as a placeholder for backend data. In a full application, this would be replaced with API-fetched data, making the component reactive to live updates and selections.
- The map provides spatial awareness to the user, supporting other components like transport option listings or route recommendations that may be rendered alongside or below it.
- It provides real time feedback by updating the map as soon as the location is changed.

The map not only reflects user input, but also allows the user to reset the destination and map state directly, making it both reactive and interactive.
The home page I worked on has 3 main components (including the map):

1. **Map –** show current location
2. **Filters –** let user search for and filter through locations, pricing and transport options
3. **Transport List –** show available transport options

## Adherence to Styling Guidelines

The component adheres to the app's styling conventions (e.g., Tailwind-based layout and custom tile layers), maintaining visual coherence across the app.

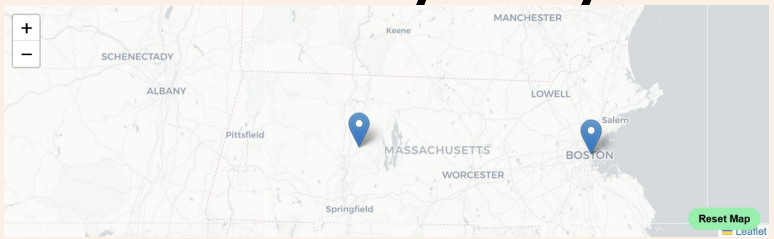
**Background & borders** match global dark theme with soft rounded edges (`rounded-xl`, `overflow-hidden`).

**Centered placement** aligns with overall layout principles (centered columnar flow).

Respects the app's **green/gray color palette** and minimalistic aesthetic

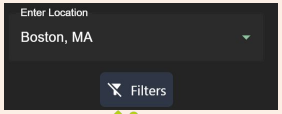
# User Flow and Component Hierarchy - Ayush

## 🔄 User Flow for the Home page:




**1. User Lands:**
User logs in to their account and reaches the home page

**2. Destination Selection:**
User selects a destination from the dropdown and uses filters → updates shared state.


**3. Map Update:**
`Map.tsx` receives new destination → finds coordinates → displays marker and fits view to show UMass + destination.

**4. Data Sync:**
Transport list and other UI components update based on selected location.

**5. Reset Interaction:**
User clicks “Reset Map” → map recenters on UMass → destination clears → filters and listings reset.


**6. UX Outcome:**
Smooth, reactive updates across map, filters, and listings → intuitive spatial navigation and transport discovery.

## 🧩 Component Hierarchy of the Map

- Part of the `Home` page layout
- Rendered alongside `Filters` and `TransportOptions` components
- Uses filters and destination props passed from Home
- App

```
└── Home
    ├── Map (destination passed from Home)
    │     └── Reset Button
    ├── Autocomplete
    ├── Filters
    └── Submit
```

# Team Collaboration and Challenges - Ayush

**Team Collaboration Takeaways**

After working as a team, I gained experience aligning component development with shared design and state management patterns based on our guidelines. I also improved communication through consistent GitHub issue tracking, standups, and code reviews. I also learnt how to manage merge conflicts when multiple people are working on different features in a codebase.

**Challenges & Solutions**

One of the main issues I faced in the development of the map is being able to dynamically add pins and scale the map to showcase both origin and destination pins. It turns out that you cannot edit the leaflet map's bounds outside of the MapContainer so I had to make another sub component that is embedded inside the main container. I also had to store a reference to the map outside useEffect in order to edit the bounds because there is no other variable to access it outside. Lastly, I had some trouble with resetting the map without unmounting the component. But I was able to add a button that allowed a smooth reset without reloading the component.

# Future Improvements and Next Steps - Ayush

## 🔧 Technical Enhancements

- **Interactive Map Features**
  Allow users to click the map to drop a marker and auto-fill the nearest destination in filters.
- **Live Data Integration**
  Replace mock data with real-time APIs for transport and location data (eg: APIs for Flights, Uber, Bus).
- **Robust Validation & Logging**
  - Use Zod for safe, schema-validated API data.
  - Integrate pino for fast and structured backend logging.
- **Better State Management**
  Explore tools like Zustand for scalable state handling.

## 🤝 Team & Process

- **Code Review Culture**
  Formalize PR reviews for more consistent code quality and shared learning.
- **User Feedback Loop**
  Plan user testing sessions to identify real-world UX pain points and gather insights on how we can make the UI more accessible.
- **Documentation & Onboarding**
  Improve docs to show how to setup the project and allow easy contribution by others.

# Assigned Work Summary - Ian

🔧 **Closed Issues:**

- **#1** – Create Community Main Page Layout (https://github.com/tanushsavadi/travy/issues/10)
- **#2** – Create "Create Post" page (https://github.com/tanushsavadi/travy/issues/9)
- **#3** – Create Listed Post component (https://github.com/tanushsavadi/travy/issues/11)

🔨 **Key Contributions:**

- Created styling and reusable input components for CreatePost component
- ListedPost component for Community page, with a limit of three lines on body text.
- Worked on Community page styling with Ipsita, combining our ideas into a single layout.
- Created reusable components for different input types.

✅ **Pull Request:**
**#20 - Community page layout**: https://github.com/tanushsavadi/travy/pull/20

# Code - Ian

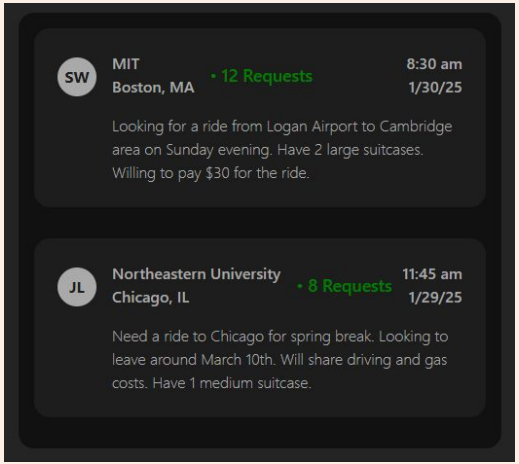
## Code for UI

```tsx
const ListedPost: React.FC<ListedPostProps> = ({ post }) => {
  const formattedDate = post.timestamp.toLocaleDateString("en-US", {…
  });

  const formattedTime = post.timestamp
    .toLocaleTimeString("en-US", {…
    })
    .toLowerCase();

  const initials = post.user.name
    .split(" ")
    .map((n) => n[0])
    .join("")
    .toUpperCase();

  return (
    <div style={ListedPostContainerCSS}>
      <div style={UserAvatar}>
        {post.user.avatar ? (
          <img
            src={post.user.avatar}
            alt={post.user.name}
            style={{ width: "100%", height: "100%", borderRadius: "50%" }}
          />
        ) : (
          <span>{initials}</span>
        )}
      </div>

      <div
        style={{
          flex: 1,
          display: "flex",
          flexDirection: "column",
          gap: "1em",
        }}
      >
        <div
          style={{
            display: "flex",
            justifyContent: "space-between",
```

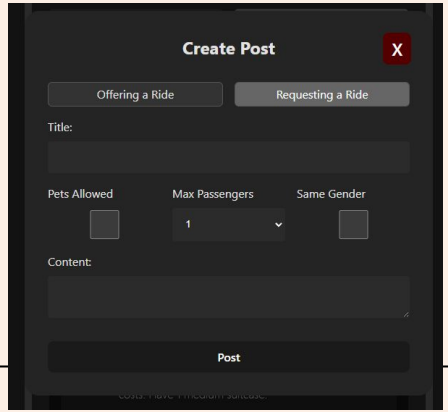
```tsx
const CreatePost: React.FC<CreatePostProps> = ({ onClose, activeTab }) => {
  const [rideType, setRideType] = React.useState<"offer" | "request">("offer");

  // Update rideType based on activeTab      ibhattacharj, 3 weeks ago • Created Communit
  useEffect(() => {…
  }, [activeTab]);

  return (
    <div className="overlay">
      <div className="content">
        <button className="close-button" onClick={onClose}>
          X
        </button>

        <h2 style={{ marginTop: "0", marginBottom: "1.25em" }}>Create Post</h2>

        <form className="form">
          <div className="offer-request-container">
            <input
              type="radio"
              id="offer"
              name="ride-type"
              checked={rideType === "offer"}
              onChange={() => setRideType("offer")}
              className="ride-type-input"
            />
            <label className="offer-request-tab" htmlFor="offer">
              Offering a Ride
            </label>
            <input
              type="radio"
              id="request"
              name="ride-type"
              checked={rideType === "request"}
              onChange={() => setRideType("request")}
              className="ride-type-input"
            />
            <label className="offer-request-tab" htmlFor="request">
              Requesting a Ride
            </label>
          </div>

          <InputField label="Title:" type="text" required />
```

## UI Result

ListedPost




CreatePost



# UI Explanation - Ian

## How This Integrates into the Overall UI Architecture

ListedPost Component

The ListedPost component is a component that displays individual post items within the community tabs:

- Data Flow: Receives post data as props from the SimpleTabs component
- Rendering Context: Rendered as a list item within the tab-content section of SimpleTabs
- Data Source: Post data comes from mockPosts
- Responsibility: Purely presentational - formats and displays post information including user details, timestamp, destination, and content

CreatePost Component

The CreatePost component is a modal form component:

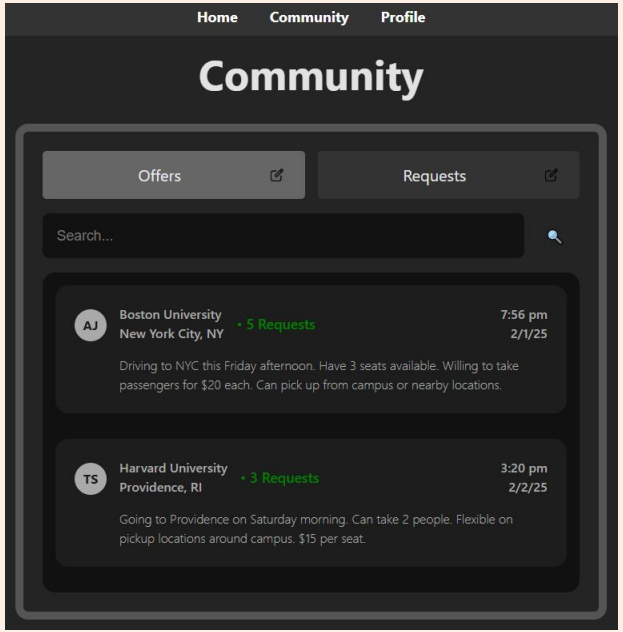
- Rendering Context: Conditionally rendered by SimpleTabs when showCreatePost state is true
- Activation: Triggered by clicking the createPostIcon in tab headers
- Props Integration: Receives onClose callback to hide the modal and activeTab to pre-select the appropriate ride type (offer || request)
- Form Composition: Uses common form components like InputField, SelectField, TextAreaField
- State Management: Tracks its own local state for ride type, synchronizing with parent's active tab

Both components work together to create a cohesive community interface where users can view existing posts (ListedPost) and create new ones (CreatePost), all within the tabbed interface provided by SimpleTabs.

# Component Hierarchy & Interaction - Ian

## 🧩 Component Hierarchy & Interaction

**User flow for ListedPost:**
App → Community page → Choose Tab → ListedPost




**User flow for CreatePost:**
App → Community page → See Tabs → CreatePost (conditional) → Form

# Challenges & Insights - Ian

One of the main challenges I had was creating reusable input components that I could use in CreatePost and other components too. Due to different styling needs, I had to balance modularity with simplicity.

Merging the work that Ipsita and I did separately on the Community page proved to be an nice challenge, as we implemented the separate views in different ways. It was challenging combining the two into one cohesive flow, but was ultimately rewarding.

Implementing text-truncation into the ListedPost contents was a learning moment, as I am not very familiar with many Webkit-specific properties.

# Future Improvements & Next Steps - Ian

**Tags Feature in Posts:** Add predefined tags that users can select from to add to their posts in the Community section, such as Uber, Lyft, Carpool, etc.

**Post Storage:** Implement a secure database for adding and storing users' posts in a persistent manner.

**Reply/Message System:** Give users a way to reply to posts and connect with other users through a messaging service.

# Assigned Work Summary - Ipsita

🔧 **Closed Issues:**

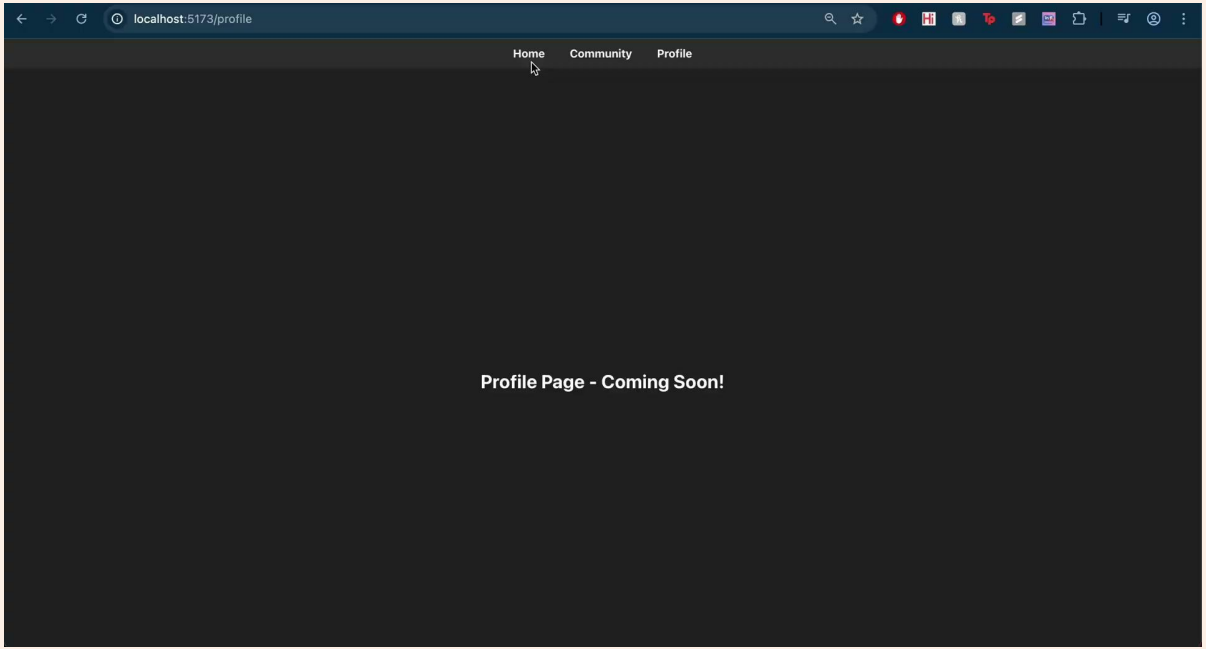
- **#1 -** Create Community Main Page Layout (#10)
- **#2 -** Producing Mock Data to Test Functionality (#14)
- **#3** – Create the Offers and Requests tab for Community page (#15)
- **#4** – Tie Pages Together (#18)
- **#5** – Component Documentation (#22)

🔨 **Key Contributions:**

- Documented all major UI components in a markdown file on repo
- Combined Ian's "Create Post" modal with the Community page outline I created; selection on two-tabbed container is dynamically reflected in modal
- Created navigation bar and incorporated react router to tie everyone's pages together
- Created mock data for User posts on Community page

✅ **Pull Request:**
**#20 - Community page layout**: https://github.com/tanushsavadi/travy/pull/20

# Screenshots & Demonstration - Ipsita




Big Takeaways

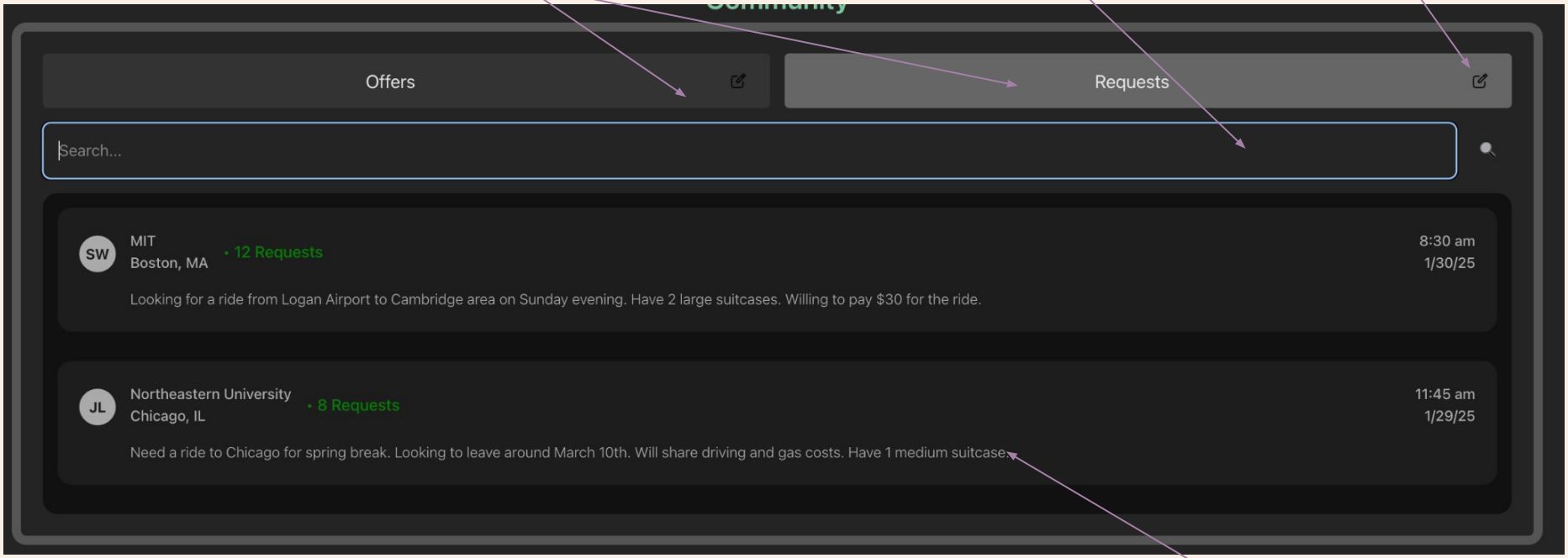
- Easy navigation between pages
- Two-tabbed community page posts; “create post” modal reflects active tab

# Screenshots & Demonstration - Ipsita

Two-tabbed container

Added search bar

Added button; connected to "create post" modal

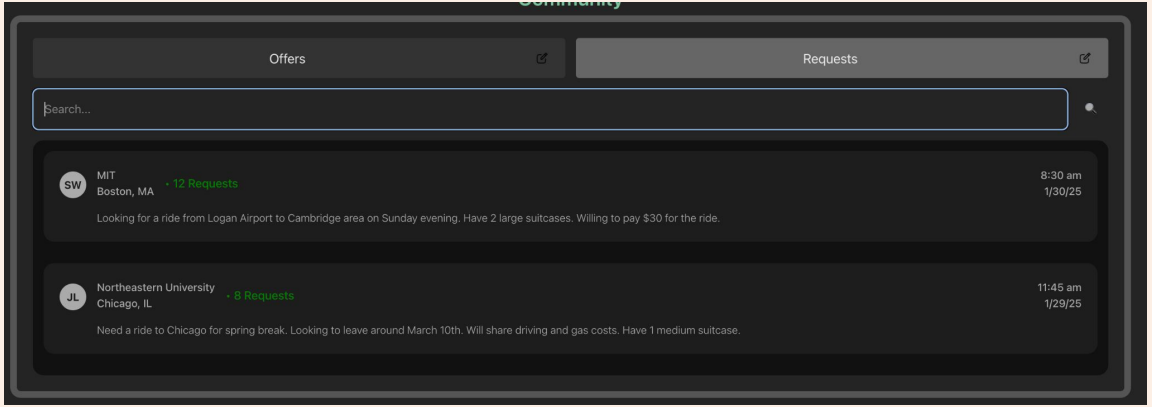



Mock post data

Created whole layout of community page

# Code & UI Explanation - Ipsita

```tsx
const SimpleTabs: React.FC = () => {
  const [activeTab, setActiveTab] = useState<"offers" | "requests">("offers");
  const [showCreatePost, setShowCreatePost] = useState(false);
  const [searchQuery, setSearchQuery] = useState("");

  const handleShowCreatePost = () => {
    setShowCreatePost(!showCreatePost);
  };

  // Filter posts based on active tab and search query
  const filteredPosts = mockPosts
    .filter((post) =>
      activeTab === "offers" ? post.type === "offer" : post.type === "request"
    )
    .filter(
      (post) =>
        post.destination.toLowerCase().includes(searchQuery.toLowerCase()) ||
        post.user.university
          .toLowerCase()
          .includes(searchQuery.toLowerCase()) ||
        post.content.toLowerCase().includes(searchQuery.toLowerCase())
    );

  return (
    <div className="tabs-container">
      <div className="tabs-header">
        <div
          className={`tab ${activeTab === "offers" ? "active" : ""}`}
          onClick={() => setActiveTab("offers")}
        >
          <span>Offers</span>
          <button
            className="create-post-button"
            onClick={(e) => {
              e.stopPropagation();
              handleShowCreatePost();
            }}
            aria-label="Create post"
          >
            <img
              src={createPostIcon}
              alt="Create Post"
              className="create-post-icon"
            />
          </button>
        </div>
        <div
```



# Code & UI Explanation - Ipsita

**Integration into UI Architecture**

- The SimpleTabs component is the main control center for the Community page, letting users switch between "Offers" and "Requests" tabs and filter posts using the search feature
- The component uses state management to keep track of which tab is active and what users are searching for, filtering posts in two ways: by type (offer or request), then via keywords/tags
- The interface shows the active tab with Travy's main color and includes a floating "+" button that opens the CreatePostModal when clicked
- Within the app's structure, the CommunityPage provides post data to SimpleTabs, which then handles filtering and works with both the PostList and create-post features
- Data Flow
    - API/State → CommunityPage → SimpleTabs (filtering) → PostList
    - SimpleTabs → CreatePostModal (user-generated content)

**Challenges & Solutions**

- Filtering for specific tabs (using if/then logic)
- Preventing unwanted click events (using stopPropagation)
- On mobile devices, the design adjusts by stacking tabs vertically, making it easy to use on any screen size

**Style Guideline Adherence**

- Uses Travy's dark theme for active tabs with System UI sans-serif typography and consistent spacing patterns from Tailwind (p-4, gap-4).
- Features keyboard-navigable tabs with proper ARIA attributes, while maintaining Material Design principles for the floating air button

# Component Hierarchy & Interaction - Ipsita

The SimpleTabs component → control center for Travy's Community page, positioned between the CommunityPage (parent) and two key child components:

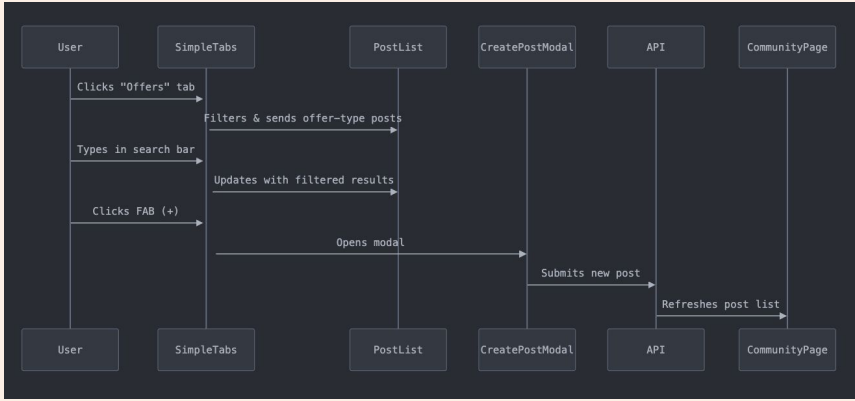
- PostList: Receives filtered posts based on active tab/search
- CreatePostModal: Triggered via the tab's floating action button (FAB)

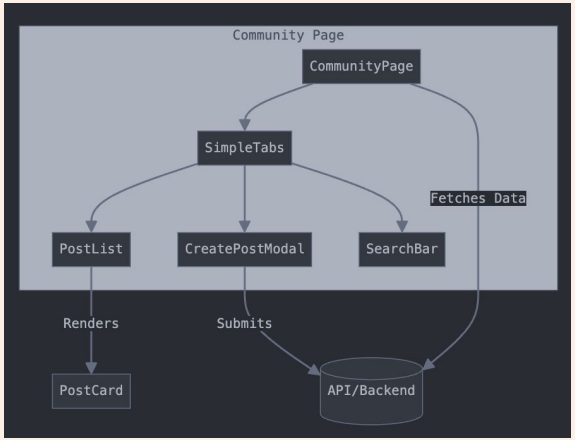
Data flows downward from CommunityPage through SimpleTabs (which handles filtering) before reaching PostList for rendering. This modular structure isolates filtering logic while enabling dynamic UI updates

---

User actions cascade through the component chain:

- Tab Selection: Switching tabs filters posts by type ("Offers"/"Requests") and instantly updates PostList
- Search: Typing in the search bar further refines the displayed posts in real time (yet to implement)
- Post Creation: Clicking the FAB opens CreatePostModal, and successful submissions refresh data via CommunityPage. Each interaction adheres to Travy's responsive and accessible design standards, ensuring a fluid user experience



# Challenges and Insights - Ipsita

I worked on the community page with Ian. We collaborated on separate portions of the same components, so trying to keep out of each other's way while ensuring that our work would be able to combine seamlessly later was a challenge. That too while the style guide hadn't been developed properly yet. I didn't know how to deal with merge conflicts before, but I learned how to in this situation.

I learned how to create "skeletal structures" in advance to seamlessly combine components later; I created the navbar and set the react router to make all pages navigable and connected.

I faced a learning curve when it came to having components dynamically updates.

# Future Improvements & Next Steps - Ipsita

- Dynamic Search
  - Implement real-time post filtering in the search bar (by destination, user, or content) with debouncing for performance.
- Database Integration
  - Set up a backend (Firestore/PostgreSQL) to store/retrieve user posts.
  - Connect the CreatePostModal to submit data to the database (via API endpoints).
  - Fetch and display posts dynamically (replace mockPosts with live data).
- Have users be able to get contact info of other users (one-way with verification)
- Add a "Load More" pagination or infinite scroll for long post lists.

Immediate Next Steps

- Build the API (e.g., Firebase/Node.js endpoints) for CRUD operations on posts.
- Connect Search Bar to filter live data (not just mockPosts).