
Minuteman Meals

Team Awesome Goated

More Meals, Less Waste

Manya, Sam, Victor, Aarav, Manu

April 4, 2025



Team Section

MINUTEMAN MEALS

A platform that helps connect dining halls or events/people with surpluses of food to students suffering from food insecurity. Our app aims to bring a streamlined solution that helps reduce food waste and food insecurity.

Key functionalities include posting free food opportunities, searching for food listings, sorting and filtering food listings based on preferences, and different profiles for users and organizations.

Repository: <https://github.com/vc64/CS426>

Milestone and issues: <https://github.com/vc64/CS426/milestone/1>

Team Members & Contributions

Manya

- Created user profile page and context
- Adjusted CSS styling (colors, fonts) and responsive page layout
- Worked on UI/UX document
- Created logo and banner

Sam

- Developed base for food card component
- Created filtering functionality
- Created mock data for food cards
- Filled out sprint report for week 3/24-3/31

Manu

- Designed final version of food cards
- Created sorting logic for default view of cards
- Implemented favorite icon for users (and implemented dynamic sorting based on state)

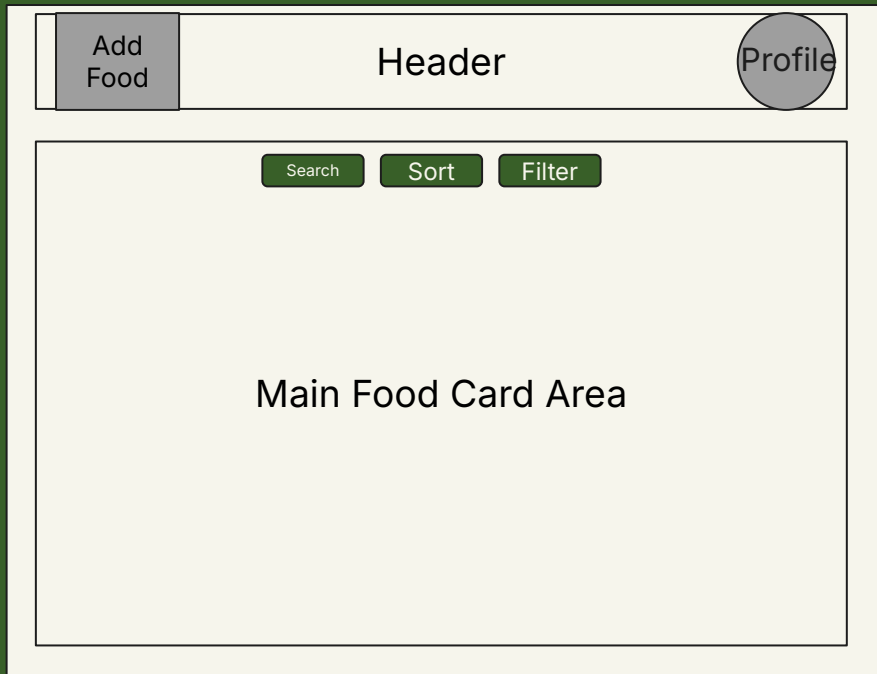
Victor

- Created functionality for organizations to add new food listings
- Helped make application more responsive for all devices
- Adapted food data to be modifiable and accessible across components

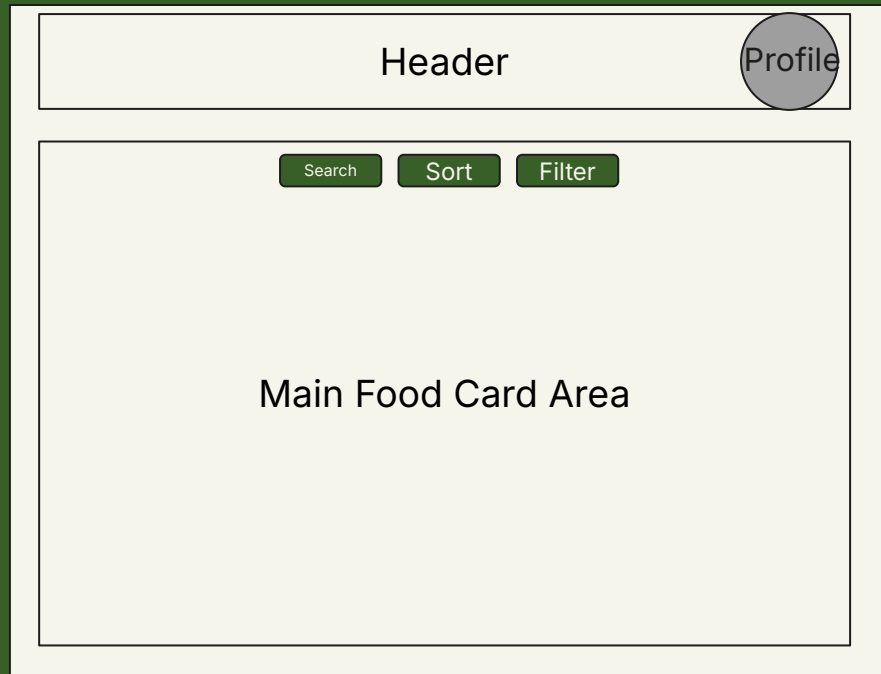
Aarav

- Created profile page and context for organizations
- Created modified version of food card for editing listings.
- Made performance checklist
- Made helper components to reduce code duplication.

Software Architecture Overview

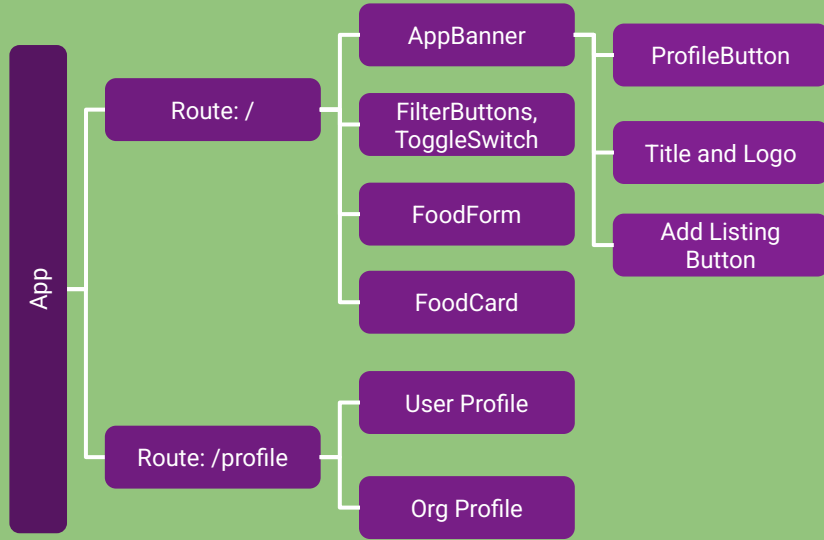


Organization View



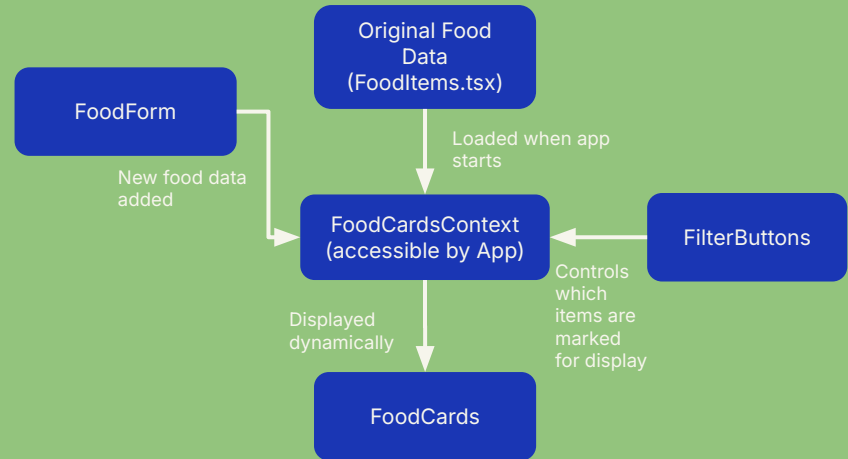
User View

Software Architecture Overview



UI Architecture

Data Flow Chart



Historical Timeline

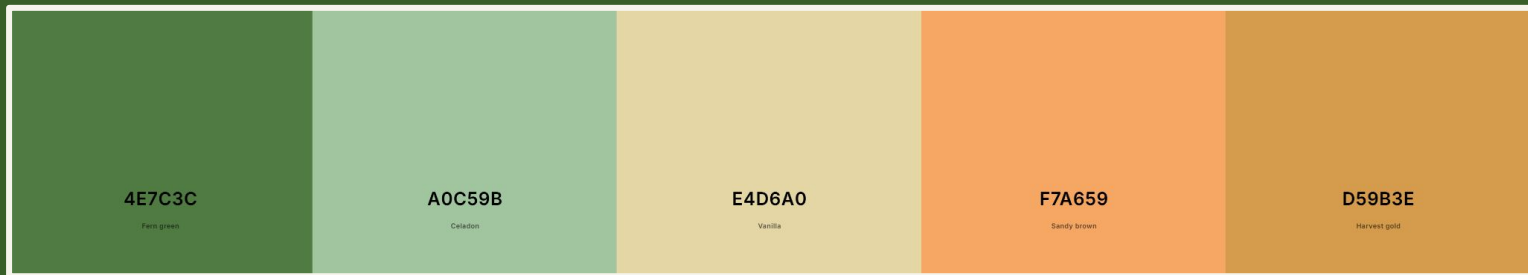
	Feb 16-22	Feb 23-Mar 1	Mar 2-8	Mar 9-15	Mar 16-22	Mar 23-29	Mar 30-Apr 5
Initial wireframing							
Component development							
Styling integration							
Integration of dynamic features							
User testing							
Final refinements							

Issues completed: Create food listings JSON mock data structure (#1), Apply styling framework (#5), Create search bar (#7), Create filtering (#8), Create sorting (#9), Choose font, font size, final color theme, and finalize other styling decisions (#11), Decide on styling frameworks (#12), Create food card component (#13), Create reusable profile component (#15), Add food listing page (#16), Use the same icons throughout application (#20), Create reusable banner (#22), Design app icon (#23), Allow user to favorite certain food cards (#24), Make home page (#25)

Design and Styling Guidelines

For fonts, Chonburi is used for Heading 1 elements, while Inter is used for all other text.

Color scheme is defined to generally be centered around green and cream colors, with the specific palette being: fern (#4e7c3c), sage (#a0c59b), cream (#e4d6a0), orange (#f7a659), and gold (#d59b3e). Color scheme is chosen to be high contrast, making it WGAC compliant. Text colors are also chosen carefully to ensure readability.



Overall, minimal transitions are used but many buttons feature slight hover effects to better indicate that these are clickable buttons, in addition to icons or text that help to describe the purpose or effect of the button.

General layout and spacing are implemented using flexbox and padding when appropriate, allowing for elements to remain evenly spaced and not squished, even when the screen is resized.

More details about styling can be found on the styling document: [426 Style Doc - Google Docs](#)

Component Documentation

Banner:



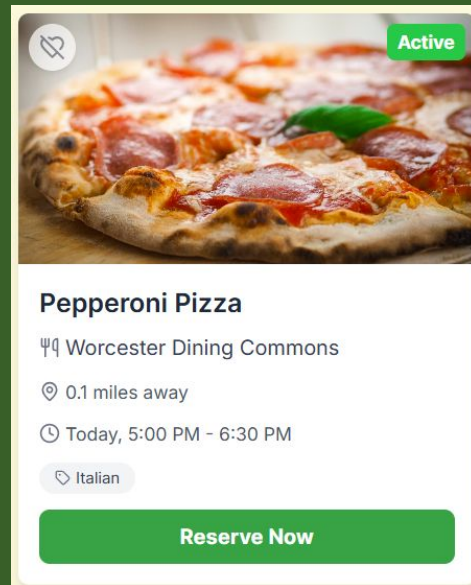
FilterButtons:



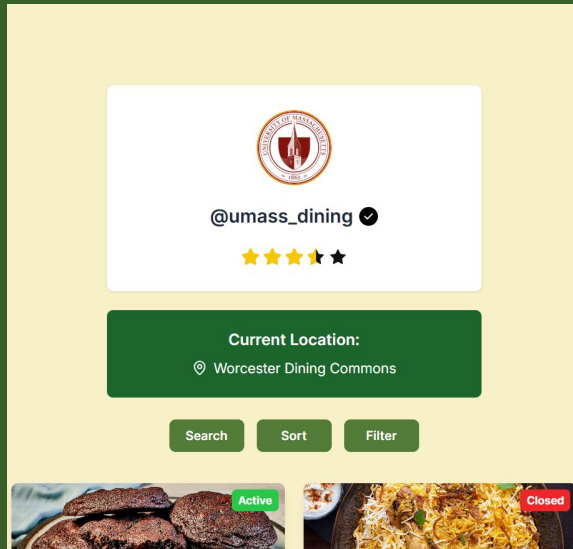
ToggleSwitch:



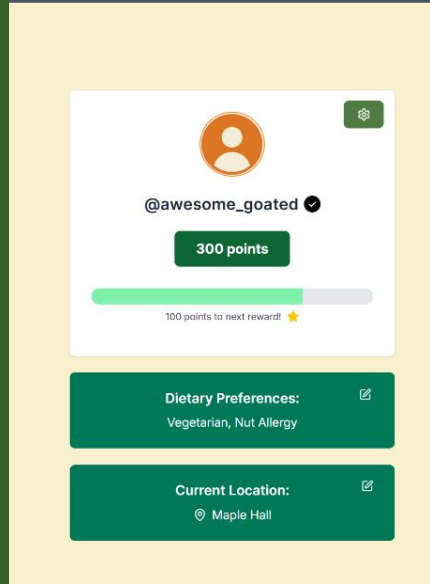
FoodCard:



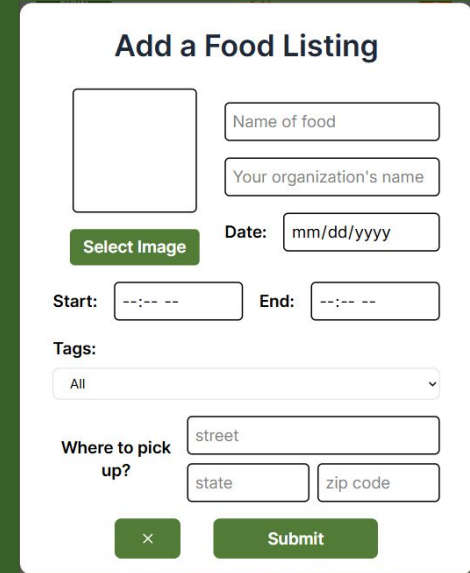
Component Documentation



OrgProfile



UserProfile



The FoodForm component is a white form titled 'Add a Food Listing'. It contains a 'Name of food' input field, a 'Your organization's name' input field, a 'Date' input field with a placeholder 'mm/dd/yyyy', and a 'Select Image' button. Below these are 'Start' and 'End' date pickers. A 'Tags' section features a dropdown menu currently set to 'All'. The 'Where to pick up?' section includes input fields for 'street', 'state', and 'zip code'. At the bottom, there is a close button (X) and a 'Submit' button.

FoodForm

Component Documentation

Components and Props/States

Banner:

- Displays the main app logo and title, as well as the profile icon. In the organization view, there is also a button for adding food listings.
- For props, the banner requires the paths to the logo and profile assets, the title and subtitle to be displayed, and whether the current viewer is an organization. It also uses the FoodListingContext to set the event handler of the add food listing button in the header.

FilterButtons:

- Features three buttons for customizing the food items displayed on the app, including search, sort, and filter.
- No props are used, but the TagContext is used in order to change the currently selected tag.

ToggleSwitch:

- Button for switching between user and organization views.
- Props include the current value (toggled or not), the function to call when the switch is toggled, and the labels to display for when the switch is on/off.

FoodCard:

- Displays all relevant information corresponding to a food item. Also features buttons for favoriting the item and reserving the item.
- Props include the food item to be displayed and a function to call when the favorite button is clicked.

Component Documentation

Components and Props/States (cont.)

FoodForm:

- Features a variety of questions for adding a food item and all of its relevant information, such as pick up time, pick up location, and a picture of the food item.
- The FoodForm does not require any props, but it does use multiple states to store the various form data that can be entered when creating a new food item. The FoodForm also uses the FoodListingContext to reset itself when the form is opened and let other components know when the form has been closed. Lastly, the FoodForm uses the FoodCardsContext to add the new food item to the list of food items.

UserProfile:

- Depicts the profile information for the current user, including their username, how many points they have, their dietary preferences, and their current location.
- The UserProfile does not require any props, but it does use the UserContext to obtain information about the current user that it needs to display.

OrgProfile:

- Depicts the profile information for the current organization, including their name, their rating, their location, and all food items posted by them.
- The OrgProfile does not require any props, but it does use the OrgContext to retrieve information about the current organization and the food items they have posted about.

Component Documentation

Contexts

TagContext:

- Stores the currently selected tag by the filter functionality. Used by the FilterButton for setting the current selected tag and used by the App for deciding which FoodCards to render.

FoodListingContext:

- Stores the open state of the add food listing pop up. Used by the App to determine when to dim the rest of the application, and used by the FoodForm to determine when to display the form.

FoodCardsContext:

- Stores all current food items and which ones are being displayed. Used by the App to determine which FoodCards to display, and used by the FoodForm to store new food items.

orgContext:

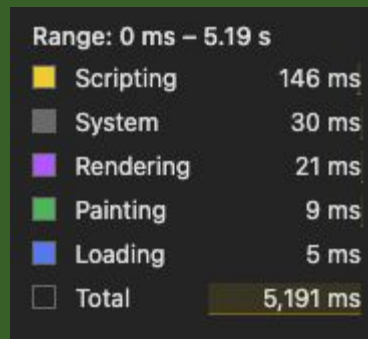
- Stores organization specific information. Used by the OrgProfile to display relevant information about the current organization on the profile page.

userContext:

- Stores user specific information. Used by the UserProfile to display relevant information about the current user on the profile page.

Performance Considerations

- Our biggest bottlenecks are “Scripting” and React, but we have taken efforts to minimize this by minimizing code duplication.
 - A component only needs to be rendered once and can be used in multiple places across the website
- React hooks are used to prevent unnecessary re-rendering



- Our only imported assets are the photos of the food, but this is necessary for our app.

Self time	Total time	Activity
3.5 ms 1.9 %	83.2 ms 45.3 %	Function call
5.2 ms 2.8 %	72.9 ms 39.7 %	performWorkUntilDeadline
37.9 ms 20.6 %	66.3 ms 36.1 %	Run console task
7.7 ms 4.2 %	39.1 ms 21.3 %	Evaluate script
36.0 ms 19.6 %	36.9 ms 20.1 %	Compile code
5.4 ms 2.9 %	25.7 ms 14.0 %	Parse HTML
1.8 ms 1.0 %	15.2 ms 8.2 %	Compile script
14.7 ms 8.0 %	14.7 ms 8.0 %	Layout
0.5 ms 0.3 %	14.7 ms 8.0 %	Evaluate module
8.7 ms 4.7 %	8.7 ms 4.7 %	(anonymous)
3.0 ms 1.6 %	8.6 ms 4.7 %	Run microtasks
1.2 ms 0.7 %	7.9 ms 4.3 %	renderWithHooks
5.5 ms 3.0 %	7.3 ms 4.0 %	exports.jsxDEV
1.6 ms 0.9 %	6.7 ms 3.6 %	(anonymous)
6.2 ms 3.4 %	6.6 ms 3.6 %	(anonymous)
2.5 ms 1.4 %	6.0 ms 3.2 %	(anonymous)

Individual Slides

Manya Phutela

Email: mphutela@umass.edu

Github: [manyap062](https://github.com/manyap062)

Assigned Work Summary

Assigned issues:

- Apply styling framework (#5)
- Choose font, size, color (#11)
- Create reusable user profile component (#15)
- Choose and implement consistent icons (#20)
- Create reusable banner (#22)
- Design app icon (#23)

PRs:

- [Implement reusable banner and icon \(#20, 22, 23\)](#)
- [Implement user profile page \(#15\)](#)
- [Styling fixes- framework, colors, fonts \(#5, 11\)](#)

✓ Design app icon

#23 · by vc64 was closed 3 weeks ago · 📌 Milestone 1 – Fr...

✓ Create Reusable Banner 1 / 1

#22 · by aarav-nair was closed last week

✓ Use the same icons throughout application (mobile and web)

#20 · by mnhegde was closed 3 days ago · 📌 Milestone 1 – Fr...

✓ Create reusable profile component 1 / 1

#15 · by aarav-nair was closed 3 days ago · 📌 Milestone 1 – Fr...

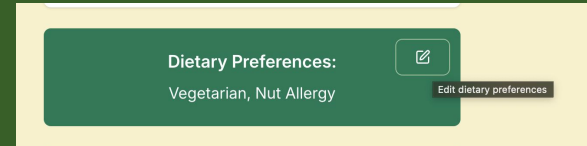
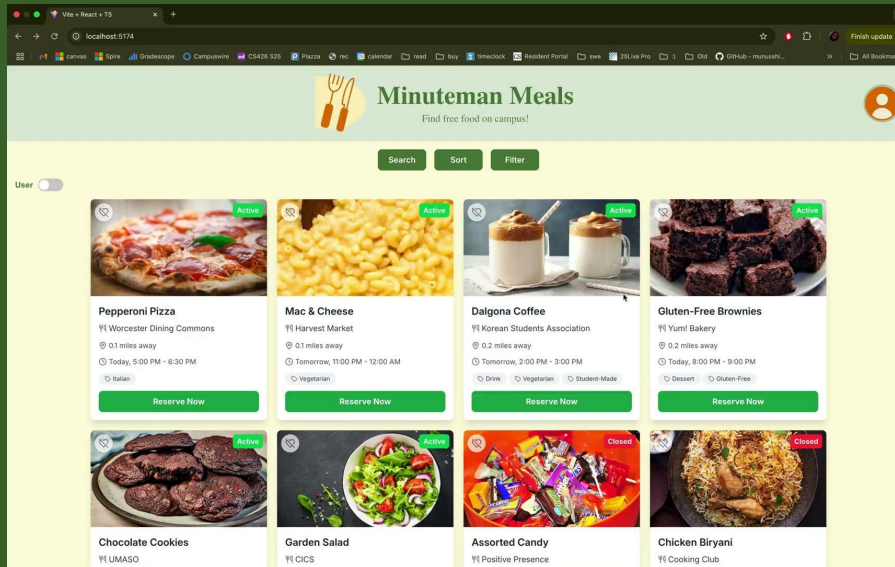
✓ Choose font, font size, final color theme, and finalize other styling decisions

#11 · by vc64 was closed 3 weeks ago · 📌 Milestone 1 – Fr...

✓ Apply styling framework 2 / 2

#5 · by sam-s04 was closed 3 weeks ago · 📌 Milestone 1 – Fr...

Screenshots and Demo



Code/UI Explanation

```
type BannerProps = {
  logoSrc: string;
  name: string;
  desc: string;
  profileSrc: string;
  isOrg: boolean
};

const AppBanner = ({
  logoSrc = "/src/assets/logo.png",
  name = "Minuteman Meals",
  desc = "Find free food on campus",
  profileSrc = "/src/assets/profile.png",
  isOrg
}: BannerProps) => {
  const navigate = useNavigate();

  const handleProfileClick = () => {
    navigate("/profile");
  };
  const foodListingContext = useContext(FoodListingContext);
  return (
    <header className="app-banner">
      <div className="banner-content">
        {isOrg && <button onClick={foodListingContext.toggleOpen}>
          New <br></br>Listing
        </button>}
        {!isOrg && <div></div>}

        <div className="logo-container justify-center">
          {logoSrc && <img src={logoSrc} alt="Logo" className="app-logo" />}
          <div className="title-container">
            <h1 className="app-title">{name}</h1>
            <p className="app-description">{desc}</p>
          </div>
        </div>

        <div className="profile-container">
          {profileSrc && (
            <img
              src={profileSrc}
              alt="Profile"
              className="profile-image"
              onClick={handleProfileClick}
            />
          )}
        </div>
      </div>
    </header>
  );
};

export default AppBanner;
```

This is the code for the banner on the dashboard of the application. This banner houses the main title and logo of the application, as well as functional buttons such as the profile icon and the new listing icon (if the current user is an organization). This dynamic nature adds complexity to the header, so ensuring that we render the correct components based on the state of the user is essential

New
Listing



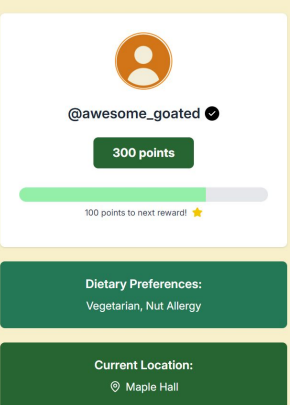
Minuteman Meals

Find free food on campus!



Code/UI Explanation

This is the code for the user view of the profile page. This profile page is rendered when the individual is a user, so it displays their username, achievements and rewards, diary preferences, location, etc. All this user information is factored into their dashboard page when it renders their recommended food cards. When we further expand the backend of the application, we can add more complexity regarding the profile state of the user



```
const UserProfile = () => {
  const { user } = useUser();

  // progress percentage for the reward bar
  const progressPercentage =
    (user.points / (user.points + user.pointsToNextReward)) * 100;

  return (
    <div
      style={{
        backgroundColor: "var(--color-cream)",
        minHeight: "180px",
        padding: "20px",
        width: "100vw"
      }}
    >
      (" ")
      <ProfileBackButton/>

      {" profile content - using proper spacing and colors to match the wireframe "}
      <div className="max-w-lg mx-auto px-4">
        {" user info row with white background "}
        <div className="bg-white rounded-lg p-8 mb-6 flex flex-col items-center text-center shadow-sm">
          <ProfileImage profileImage={user.profileImage} username={user.username} />
          <ProfileUsername username={user.username} isVerified={user.isVerified} />

          {" Points display - green bottom "}
          <div className="bg-green-800 text-white font-bold rounded-lg px-8 py-3 mb-8 text-xl inline-block">
            300 points
          </div>

          {" Progress bar - matching the green gradient in wireframe "}
          <div className="w-full max-w-md mb-2">
            <div className="h-6 bg-gray-200 rounded-full overflow-hidden">
              <div
                className="h-full bg-green-300"
                style={{ width: `${progressPercentage}%` }}
              </div>
            </div>

            {" Points to next reward with yellow star "}
            <div className="flex items-center justify-center text-sm mb-4 text-gray-700">
              <span>{user.pointsToNextReward} points to next reward</span>
              <Star
                className="ml-2 text-yellow-400"
                size={18}
                fill="currentColor"
              />
            </div>

            {" Dietary preferences section - light green background "}
            <div className="bg-emerald-700 p-6 rounded-lg mb-6">
              <div className="text-xl font-bold mb-2 text-center text-white">
                Dietary Preferences:
              </div>
              <p className="text-lg text-center text-white">
                {user.dietaryPreferences.join(", ")}
              </p>
            </div>

            <ProfileLocation currentLocation={user.currentLocation}/>

          </div>
        </div>
      </div>
    </div>
  );
};

export default UserProfile;
```

Challenges and Insights

Challenges

- Ensuring banner looked good and was responsive on all screen sizes
 - Making sure this remained when it was changed based on individual (user vs org)
 - When toggled to organization, we needed an "Add Listing" button rendered as well
- Emulate existing user profiles
 - Clearly and easily display information to the user
 - Make sure this is editable so that user can always change their preferences

Insights

- Ensure constant communication so that we all know what we're working on, making collaboration more effective
 - This also reduced the amount of merge conflicts and confusion on github between our different branches
- Feel much stronger about state management using React across multiple pages/components

Future Improvements

- Designing pages/pop-ups for editing profile information, including verification screen for users
- Improving on accessibility features - for example, tooltips and ensuring all form inputs have labels
- Create sign in/up and home page on open, compatible for mobile and web

Samantha Sheedy

Email: ssheedy@umass.edu

Github: [sam-s04](#)

Assigned Work Summary

Assigned issues:

- Create food card component (#13)
- Create food listings mock data (#1)
- Create user profile mock data (#2)
- Create filtering button and functionality (#8)

PRs:

- [Mock data and filtering \(#29\)](#)

Remaining issues:

- Create user profile/organization mock data (#2)
- Add additional sorting/filtering for posters to see their own listings (#10)

Committed these early in the project before reading the commit message guidelines, my bad 😞 →

Commits:

Make food card mock data, create placeholders for

search, sort, and filter buttons

- Extended mock data so that there are now eight food cards
- Made mock data more relevant to our app, which centers around UMass
- Created placeholders for search, sort, and filter buttons. Working on filtering functionality next

Resolves [#1](#), contributes to [#4](#)

 sam-s04 committed 4 days ago

feat: Filtering functionality

- Created context for the selected tag in the dropdown menu
- Changed App.tsx so that cards are filtered by selected tag, with default tag 'All'
- Wrapped the App in a TagProvider

Resolves [#8](#)

 sam-s04 committed 3 days ago

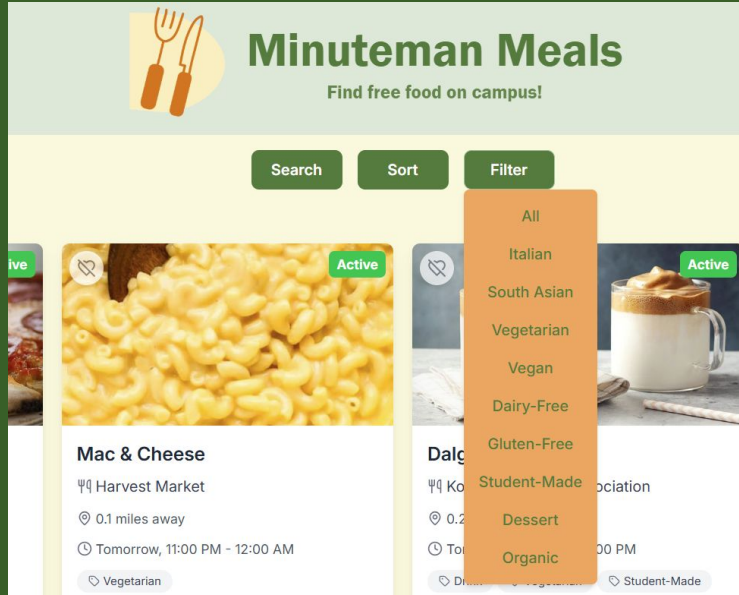
food card basics

 sam-s04 committed 2 weeks ago

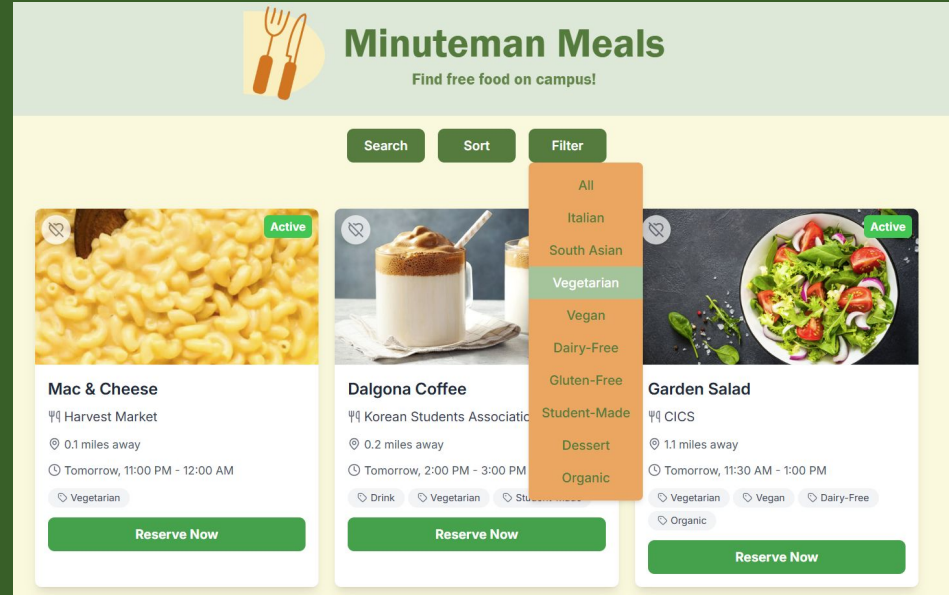
bones of food cards finished. styling needed

 sam-s04 committed 2 weeks ago

Screenshots & Demonstration



Hover over the filter button to see a list of tags, which include cultures, dietary needs/preferences, etc.



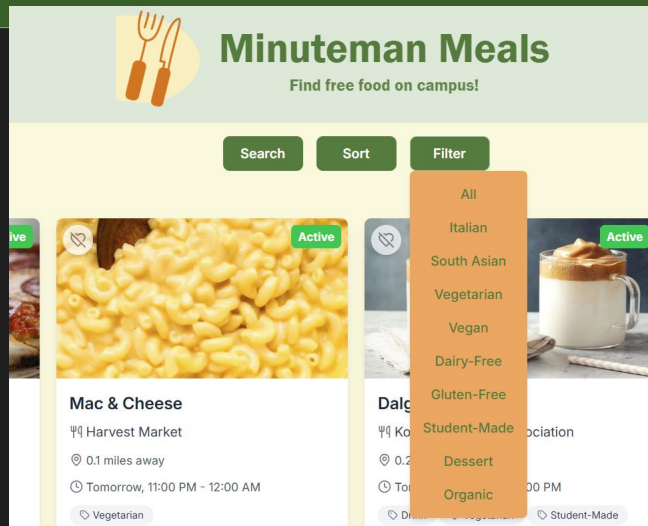
Clicking a tag from the drop-down menu will result in only food listings with that tag appearing.

Code/UI Explanation

```

1 import { useTag } from './contexts/TagContext.tsx';
2
3 const buttonStyle = {
4   backgroundColor: '#16a34a',
5   color: 'white',
6   fontWeight: 'bold',
7   padding: '0.5rem 1rem',
8   borderRadius: '0.5rem',
9   width: '5rem',
10 }
11
12 const SearchSortFilter = () => {
13
14   const tags = ['All', 'Italian', 'South Asian', 'Vegetarian', 'Vegan', 'Dairy-Free', 'Gluten-Free', 'Student-Made', 'Dessert', 'Organic']
15
16   const { setSelectedTag } = useTag();
17
18   const handleClick = (item : string) => {
19     setSelectedTag(item);
20   }
21
22   return (
23     <div className="flex justify-center gap-4 mb-4">
24       <button style={buttonStyle}>Search</button>
25       <button style={buttonStyle}>Sort</button>
26
27       /* Drop-down menu for filter button */
28       <div className="group">
29         <button style={buttonStyle}>Filter</button>
30         <ul className="py-2 grid grid-cols-1 bg-gray-200 absolute opacity-0 group-hover:opacity-100 group-hover:pointer-events-auto pointer-events-none z-10">
31           {tags.map((item) => (
32             <li key={item} className="px-4 py-2 hover:bg-gray-300 cursor-pointer" onClick={() => handleClick(item)}>{item}</li>
33           ))}
34         </ul>
35       </div>
36     </div>
37   )
38 }
39
40
41 export default SearchSortFilter;

```



Code/UI Explanation

Explanation

The code on the previous slide creates the Search, Sort, and Filter buttons. It also builds a drop-down menu of possible tags that appears under the Filter button when it is hovered over, and implements the handleClick event to set the selected tag when a menu item is clicked. This feature uses a context so that the selected tag can be used in other files, where the actual filtering of the cards takes place.

Styling

The buttons match the color scheme, and they maintain the same rounded corners and padding as the other buttons in the app.

Component interaction

The TagContext allows for selectedTag and setSelectedTag to be accessed throughout the application. FilterButton.tsx uses this context to set the selected tag to whichever tag from the dropdown menu that was clicked. Then, in App.tsx, the selected tag is accessed, and after the food cards are sorted, they are filtered by the selected tag and only those with the relevant tag are shown.

Challenges and Insights

Challenges

The biggest challenge in developing the filtering button and functionality was making sure that the drop-down menu only appears when the button is hovered over. I had gotten it so that it appeared on hover, but when the whole invisible menu was hovered over and not just the button. I also had challenges using the context.

Solutions

I fixed the menu by researching a bit about how Tailwind CSS works and using the "group-hover" tag with the button and menu within a "group" div element. For the context, I learned that all context usage must be done inside of a function, such as the arrow function for a component.

Takeaways

This milestone helped me learn more about how contexts work and how to apply them. I also gained working knowledge of Tailwind CSS. As for working in a collaborative team environment, I feel that I have developed strong technical skills in Git for version control. Working on this milestone has shown me how important communication is in combining all of the separate parts of the project and minimizing merge conflicts and errors.

Future Improvements

- Expanding upon search and sort functionalities beyond our initial implementations
- Allowing organizations to filter by listings that they posted
- Creating tooltips for effective user feedback
- Refining the User/Organization toggle

Victor Chen




Email: vqchen@umass.edu

Github: [vc64](#)

Assigned Work Summary

Commits

Assigned Issues

-  [Adapt web styling for mobile and responsive viewing \(#6\)](#)
-  [Decide on styling frameworks \(#12\)](#)
-  [Create add listing page \(#16\)](#)

PRs

- [Add food listing form \(#33\)](#)

Contributions

- My primary tasks for this milestone were to decide on a styling framework for the application and creating the functionality for organizations to post new food listings.
- I also helped with other tasks, with a main focus on ensuring other components are responsive and work cohesively together. I adjusted the FoodCard and Banner components to be more responsive using flexbox. I introduced the FoodCardsContext in order to keep track of the current food items and which ones are visible, and integrated this into the existing components. I also fixed issues that emerged with the filter buttons and sorting functionality when components were combined together.

move toggle switch to same line as the filter buttons

vc64 committed 2 days ago

remove unnecessary prop from FoodForm

vc64 committed 2 days ago

improve food form format slightly, allowing for form to fit better on screen for all devices

vc64 committed 2 days ago

make form refresh each time it is reopened - #16

vc64 committed 5 days ago

small changes to resolve type warnings

vc64 committed 5 days ago

fix filtering issue by adding useEffect in App.tsx to react to selectedTag changes, also separate list of all food items vs food items being shown

vc64 committed 5 days ago

add tag support for food listing form and perform some styling changes - Create add listing page #16

vc64 committed last week

add new context for food cards, allowing addition of new food cards - Create add listing page #16

vc64 committed last week

fix issue with two color sin background of main app, and fix width of background for profile

vc64 committed last week

adjust button width slightly to fix search button issue

vc64 committed last week

add styling to form - Create add listing page #16

vc64 committed last week

add preliminary food listing form - Create add listing page #16

vc64 committed last week

adjust foodCard width

vc64 committed last week

add context for food listing form to allow components to know whether the form is open - Create add listing page #16

vc64 committed last week

add button for food listing form - Create add listing page #16

vc64 committed last week

fix issue with background not spanning entire page

vc64 committed last week

increase responsiveness of banner - Adapt web styling for mobile and responsive viewing #6

vc64 committed last week

add button for food listing to banner, change from absolute positioning to flex - Create add listing page #16

vc64 committed last week

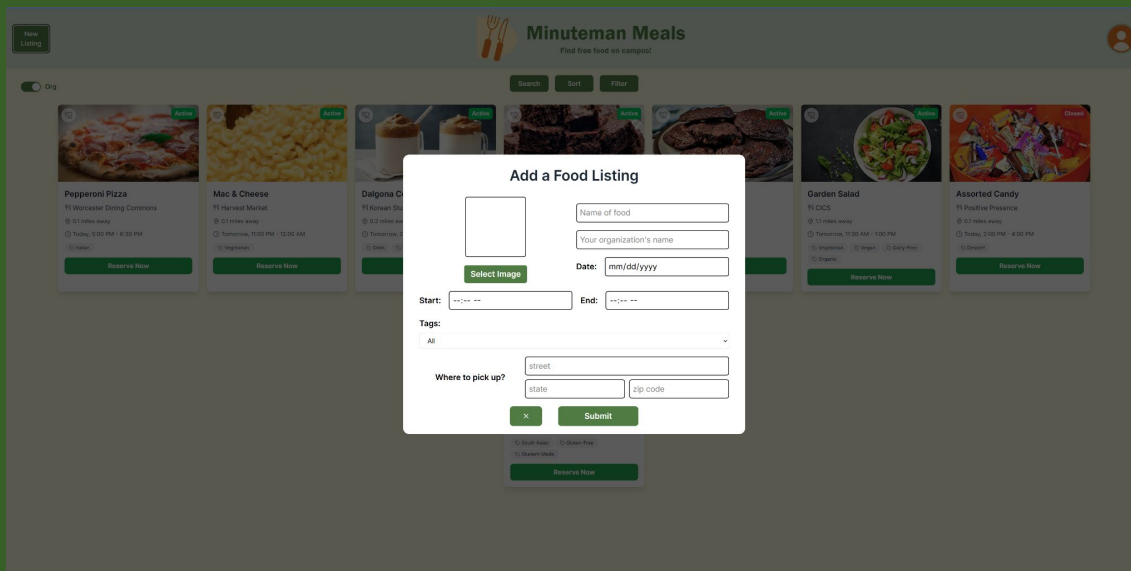
add initial food form - Create add listing page #16

vc64 committed last week

Personal Contributions

Below is the FoodForm component I implemented. When the “New Listing” button in the Banner is clicked, an overlay appears, dimming the main application. The FoodForm is also then shown in the center of the screen, featuring ways to input information such that the food name, organization name, pickup time, tags, and pickup location.

This FoodForm enables organizations to post new food items in a convenient and easy-to-use form. Through the FoodForm, new food items can be generated and stored in our application. In addition, these items can be displayed to users and organizations in the main application, with existing filters being automatically applied to them. This component is designed to draw the user's focus to the form itself, and the background dim helps to both draw attention away from the rest of the application temporarily and prevent users from interacting with the rest of the application while the FoodForm is still open.



The screenshot displays the 'Minuteman Meals' web application interface. At the top, there is a navigation bar with a 'New Listing' button on the left and a user profile icon on the right. Below the navigation bar, a search bar with 'Search', 'Sort', and 'Filter' buttons is visible. The main content area shows a grid of food items, each with a photo, name, and details. Overlaid on this grid is a modal form titled 'Add a Food Listing'. The form contains the following fields and controls:

- Select Image:** A button to upload a food image.
- Name of food:** A text input field.
- Your organization's name:** A text input field.
- Date:** A date input field with a placeholder 'mm/dd/yyyy'.
- Start:** A time input field with a placeholder 'mm-dd--'.
- End:** A time input field with a placeholder 'mm-dd--'.
- Tags:** A dropdown menu currently set to 'All'.
- Where to pick up?:** A section containing three input fields: 'street', 'state', and 'zip code'.
- Submit:** A green button to submit the form.
- X:** A small button to close the form.

Code/UI Explanation

```

90 const onSubmit = () => {
91   const newFood: foodItemType = {
92     id: 0,
93     foodName: food,
94     restaurantName: orgName,
95     imageUrl: "Pizza.jpg",
96     distance: 0,
97     pickupTime: `${times ? times.date : ""}, ${times ? times.start : ""} - ${times ? times.end : ""}`,
98     tags: Object.entries(tags).filter(([,v]) => v).map(([k,]) => k),
99     active: true,
100    isFavorite: false
101  }
102  addCard(newFood);
103  toggleOpen();
104 }

207 <div className="flex justify-center gap-10">
208   <button
209     onClick={toggleOpen}
210     className="flex justify-center items-center rounded-full bg-gray-300 hover:bg-gray-400 transition w-20">
211     <X className="w-5 h-5 text-white-700" />
212   </button>
213   <button
214     onClick={onSubmit}
215     className="flex justify-center rounded-full bg-gray-300 hover:bg-gray-400 transition w-50">
216     <p className="text-xl font-semibold text-white-700">Submit</p>
217   </button>
218 </div>

```

The buttons implemented by the code above

- This code illustrates what is used to produce the submit and exit buttons at the bottom of the FoodForm. These buttons are contained in their own div, similar to the other form items in the FoodForm.
- This code enables the functionality of creating a new food listing using the information provided by the user in the FoodForm. This results in a new FoodCard being created and displayed in the main application.
- It also uses the FoodListingContext to let other components know that the FoodForm has been closed. This results in the background overlay (which dims the application) to also disappear.
- There were some difficulties with creating a new food item, as errors emerged when the user does not provide responses to some of the form questions. To combat this, the onSubmit function checks to ensure that the times object is not null, substituting the pickupTime with an empty string if the times object is null. Another challenge that was faced was positioning the two buttons such that they also fit with the style of the entire FoodForm. Ultimately, flexbox was used in order to center the buttons in the form while also leaving ample space between them. The button colors are also chosen such that they are readable.

Challenges and Insights

Overall, while I faced a lot of obstacles when implementing my features and completing my tasks, the most challenging issues I had were centered around states and contexts. Since our application features several components that require access to specific information or need to modify certain information, it was crucial to use contexts correctly in our application. When debugging, I often found that the issues stemmed from my incorrect use of contexts, and at times these issues were difficult to resolve. However, these experiences have helped me understand contexts better and become more comfortable using them in order to synchronize information between components.

As a group, this milestone also provided some difficulties in collaboration. In particular, since we often worked on our personal tasks independently, many merge conflicts emerged when we attempted to perform PRs to combine our changes. Effective communication and collaboration, as well as patience, allowed us to overcome these challenges.

I think our application has room to improve in terms of ensuring compatibility on mobile devices. Currently, some components are not mobile-friendly, with some having too wide of a minimum width, and others relying on the mouse hover behavior that is exclusive to computers.

Manu Hegde

Email: manuhegde@umass.edu

Github: [mnhegde](https://github.com/mnhegde)

Assigned Work Summary

Assigned issues:

- Functionality for finding food cards (#4)
- Create search/sorting (#9)
- Create food card component (#13)
- Create Like and Dislike buttons (#14)
- Allow user to favorite foods (#24)

PRs:

- [Implemented Food Card Component \(#26\)](#)
- [Sorting and Display of Foods \(#28\)](#)


Remaining issues:


- Functionality for finding food (#4)
- Create tooltips (#18)
- Organizational functions on food cards (#21)

Commits:

Feat: Refactored styling for food cards


- Redesigned food cards and their styling to leverage tailwind
- Integrated features from lucide-react to creatively display information like pickup time, tags, etc.
- Made component reusable to make scalability easy


 main (#26)

 mnhegde committed 2 weeks ago

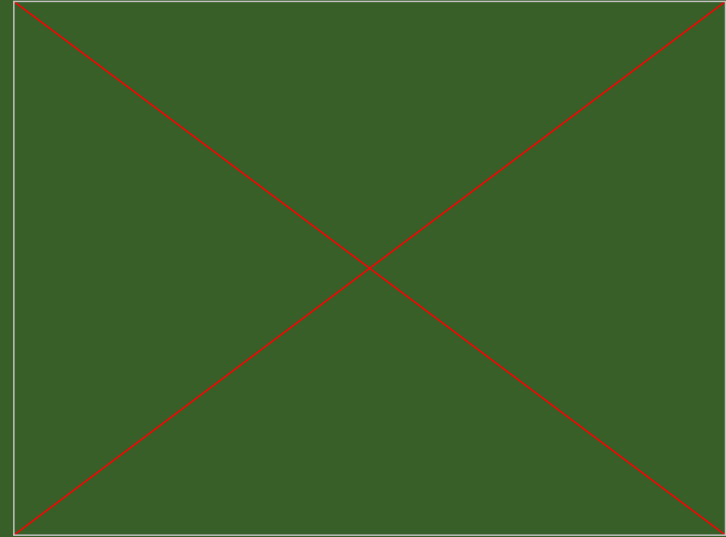
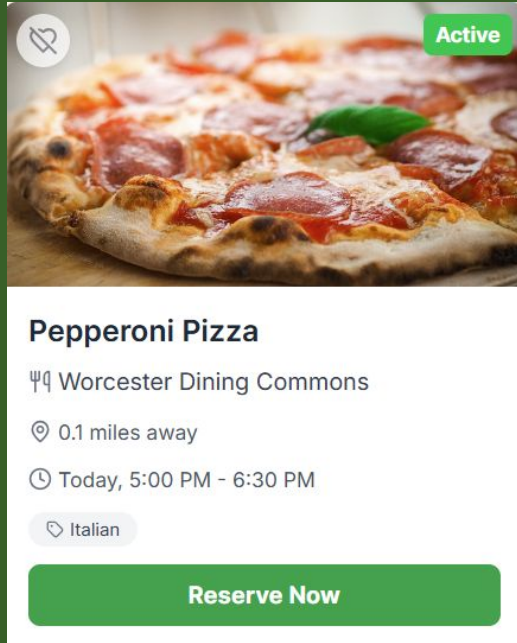
Sort food by active status and favorite by default, and dynamically s...

...orts cards when favorites are updated

 main (#28)

 mnhegde committed 2 days ago

Screenshots & Demonstration



Code/UI Explanation

```
const cardMap = new Map<number, foodItemType>();
// foodItems.forEach((e) => cardMap.set(e.id, e));
foodCards.forEach((e) => cardMap.set(e.id, e));
// We want to separate first into two groups - active or not active
// Then order first by favorites, then by distance
const sortFood = (arr: foodItemType[]) => {
  const favorites = arr.filter((e) => e.isFavorite);
  const notFavorites = arr.filter((e) => !e.isFavorite);
  favorites.sort((a, b) => a.distance - b.distance);
  notFavorites.sort((a, b) => a.distance - b.distance);
  const sortedCards = favorites.concat(notFavorites);
  // Need to implement functionality so user can choose this
  return selectedTag === "All"
    ? sortedCards
    : sortedCards.filter((item) => item.tags.includes(selectedTag));
};

// Use useEffect to set initial cards only once
useEffect(() => {
  setFoodCards(sortCards());
}, []); // Empty dependency array means this runs only once on mount

// update food cards whenever selectedTag changes
useEffect(() => {
  console.log(selectedTag);
  setFoodCards(sortCards());
}, [selectedTag, allFoodItems]);

const sortCards = () => {
  // const active = foodItems.filter((e) => e.active);
  // const notActive = foodItems.filter((e) => !e.active);
  const active = allFoodItems.filter((e) => e.active);
  const notActive = allFoodItems.filter((e) => !e.active);
  const activeList = sortFood(active);
  const notActiveList = sortFood(notActive);
  return activeList.concat(notActiveList);
};

const handleFavChange = (id: number) => {
  // const foodCard = cardMap.get(id);
  const foodCard = foodCards.find(item => item.id === id);
  foodCard!.isFavorite = !foodCard!.isFavorite;
  setFoodCards(sortCards());
};
```

This is the code that sorts and displays the food cards for the user. This makes it so that the food cards that are active are displayed first, and among these the favorites are displayed at the top. If a food card is favorited by the user, this sorting is dynamically updated and the display re-renders for the user

Code/UI Explanation

This is the code for the food card component, which encapsulates all the data for each dining location. This not only contains the UI for the food cards, but the logic for the different aspects that can be interacted with by the user. This component is essential for each of the individual food cards, so this is the information users are seeing as they scroll through the homepage of the application

```
// foodCard.tsx
const FoodCard = ({ food, favToggle }: {food: foodItemType, favToggle: (index: number) => void}) => {
  // allow us to have default values for the card
  const foodData = { ...defaultFood, ...food };

  const {
    id,
    foodName,
    restaurantName,
    imageUrl,
    distance,
    pickupTime,
    tags,
    active,
    isFavorite
  } = foodData;

  // In future, may want to re-render food cards so that those favorited pop up first...
  // So need to take this into account when mapping grid
  const handleFavoriteClick = () => favToggle(id);

  return (
    // Used many Tailwind CSS classes to have the styling applied
    <div className="max-w-sm rounded-lg overflow-hidden shadow-lg bg-white hover:shadow-xl transition-shadow duration-300 w-[350px]">
      <div className="relative">
        <img
          src={`/${src/assets/` + imageUrl`}
          alt={foodName}
          className="w-full h-48 object-cover"
        />
        {active ?
          (<div className="absolute top-0 right-0 bg-green-500 text-white px-2 py-1 m-2 rounded-md text-sm font-bold">Active</div>
          : (<div className="absolute top-0 right-0 bg-red-500 text-white px-2 py-1 m-2 rounded-md text-sm font-bold">Closed</div>)}

        <div className="absolute top-0 left-0 m-2" onClick={handleFavoriteClick}>
          {isFavorite ? (
            <div className="bg-white p-2 rounded-full shadow-md">
              <Heart size={20} color="#ef4444" fill="#ef4444" />
            </div>
          ) : (
            <div className="bg-white p-2 rounded-full shadow-md opacity-70 hover:opacity-100">
              <HeartOff size={20} color="#6b7280" />
            </div>
          )}
        </div>
      </div>
    </div>
  );
}
```

Challenges and Insights

Obstacles

- Maintaining global context - cards each maintained their own favorite state, so when this was changed the App had to be re-rendered so that the cards would be in the correct order
 - Bugs such as infinite re-renders or state not updating correctly
- Card UI - ensuring the information was easily displayed and succinct for users to effectively navigate
 - Storing this information effectively so that sorting, filtering, etc. could be done easily

Insights

- Ensuring constant communication so teammates understood what everyone was working on
 - This allowed us to split up work most effectively
 - Minimized merge conflicts that would occur between different branches
- Many moving parts that were connected, so coordinating with others so that intertwined parts could be developed together and allow for consistent user experience

Future Improvements

- Once we start on integrating the backend, continue to improve the food cards generated on the dashboard based on the profile of the user
 - More robust recommendation, maybe use a different algorithm to present this to user
- Establish the user flow of the application
 - Add better UX features like tooltips, pop ups, etc. so that application is easy to use
- Flesh out the reservation flow
 - We need to ensure that the making a reservation as a user is easy for both parties
 - Expand on how this information is stored in application and managed for high-volume events

Aarav Nair

Email: aanair@umass.edu

Github: [aarav-nair](https://github.com/aarav-nair)

Assigned Work Summary

Assigned issues:

- [Create reusable components \(#3\)](#)
- [Create messaging system \(#17\)](#)
 - We have dropped this feature for now so not completed.
- [Organization Profile View](#)
- [Home Page](#)

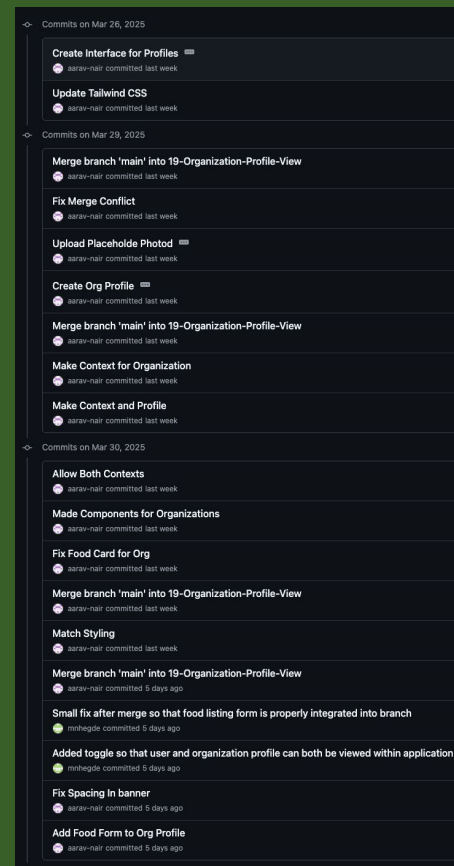
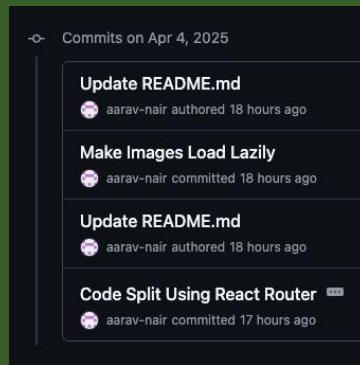
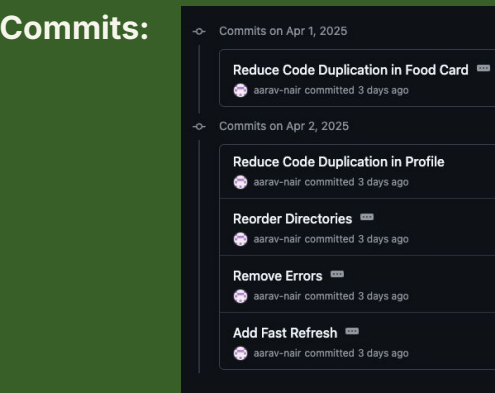
PRs:

- [Organization View \(#32\)](#)
- [Organization Editing and Tweaks \(#34\)](#)
- [Performance Checklist \(#36\)](#)

Remaining issues:

- Organizational functions on food cards (#21)
 - Didn't finish this issue in its entirety because editing issues is a backend task. Right now we have created the editing view though.

Commits:



Code/UI Explanation

This is the code with the helpers for the food card component. We ended up making two different components, one to show the users and one to show the organizations. But a lot of things, such as the image of the food, are shared so I put them into a helper component. I put some constants in a different file so we could take advantage of React Fast Refresh.

```
import { Clock, Tag, MapPin, Utils } from "lucide-react";

export const FoodCardImage = ({ imageUrl, foodName, active }: { imageUrl: string; foodName: string; active: boolean }) => {
    <div>
        <img
            loading="lazy"
            src={"/src/assets/" + imageUrl}
            alt={foodName}
            className="w-full h-48 object-cover"/>
        />
        {active ?
            (<div className="absolute top-0 right-0 bg-green-500 text-white px-2 py-1 m-2 rounded-md text-sm font-bold">Active</div>) :
            (<div className="absolute top-0 right-0 bg-red-500 text-white px-2 py-1 m-2 rounded-md text-sm font-bold">Closed</div>)}
        </div>
    };
};

export const FoodCardText = ({ foodName, restaurantName, distance, pickupTime, isUser }: { foodName: string; restaurantName: string; distance: number; pickupTime: string; isUser: boolean }) => {
    <div>
        <div className="flex justify-between items-start mb-2">
            <h3 className="text-xl font-bold text-gray-800">(foodName)</h3>
            </div>
        <div className="flex items-center mb-3">
            <div size={16} className="text-gray-500 mr-1"/>
            <span className="text-gray-700">(restaurantName)</span>
            </div>
        {isUser ?
            (<div className="flex items-center mb-3">
                <MapPin size={16} className="text-gray-500 mr-1"/>
                <span className="text-gray-600 text-sm">(distance + " miles away")</span>
            ) :
            (<div className="flex items-center mb-3">
                <Clock size={16} className="text-gray-500 mr-1"/>
                <span className="text-gray-600 text-sm">(pickupTime)</span>
            )}
        </div>
    </div>
};

export const FoodCardTags = ({ tags }: { tags: string[] }) => {
    <div className="flex flex-wrap gap-1 mb-3">
        {tags.map((tag, index) => {
            <div key={index} className="flex items-center bg-gray-100 rounded-full px-3 py-1 text-xs">
                <Tag size={12} className="text-gray-500 mr-1"/>
                <span className="text-gray-700">{tag}</span>
            </div>
        ))}
    </div>
};
```

```
export const defaultFood = {
  foodName: "Assorted Pastries",
  restaurantName: "Bella Bakery",
  imageUrl: "/api/placeholder/300/200",
  distance: 0.7,
  pickupTime: "Today, 5:00 PM - 6:30 PM",
  tags: ["Vegetarian", "Bakery"],
  active: true,
  isFavorite: true
};

export const foodCardButtonStyle = {
  backgroundColor: '#16a34a', // Tailwind green-600
  color: 'white',
  fontWeight: 'bold',
  padding: '0.5rem 1rem',
  borderRadius: '0.5rem',
  width: '100%',
  transition: 'background-color 200ms',
};

export const handleMouseEnter = (e) => e.target.style.backgroundColor = '#15803d';

export const handleMouseLeave = (e) => e.target.style.backgroundColor = '#16a34a';
```

Aarav Nair, aanair@umass.edu, [aarav-nair](#)

Code/UI Explanation

```
/* profile content - using proper spacing and colors to match the wireframe */
<div className="max-w-lg mx-auto px-4">
  /* org info card with white background */
  <div className="bg-white rounded-lg p-8 mb-6 flex flex-col items-center text-center shadow-sm">
    <ProfileImage profileImage={org.profileImage} username={org.username} />
    <ProfileUsername username={org.username} isVerified={org.isVerified} />

    /* Star Ratings */
    <div className="flex space-x-2">
      <div className="relative">
        <div className="flex gap-1">
          { Array.from({ length: 5 }, () => {
            <Star fill="#111" strokeWidth={0} />
          }) }
        </div>
        <div className="absolute top-0 flex space-x-1">
          { [...Array(Math.ceil(stars)).map((_, index) => {
            const starValue = index + 1;
            return starValue === Math.ceil(stars) ? (
              <StarHalf
                className="text-yellow-400"
                fill="currentColor"
              />
            ) : (
              <Star
                className="text-yellow-400"
                fill="currentColor"
              />
            );
          }) ] }
        </div>
      </div>
    </div>

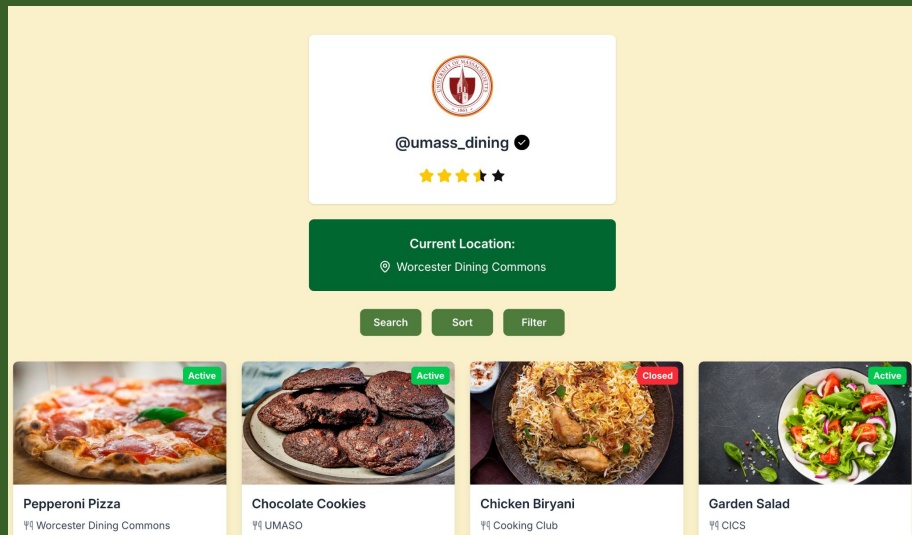
    <ProfileLocation currentLocation={org.currentLocation} />
  </div>

  /* Form for editing food */
  <div id="overlay" className={ `fixed inset-0 w-full h-full bg-black/[0.65] z-2 ${foodListingContext.isOpen ? "" : "hidden"} ` }></div>
  <div className={ `fixed inset-0 w-full h-full flex justify-center items-center z-3 ${foodListingContext.isOpen ? "" : "hidden"} ` }>
    <FoodForm></FoodForm>
  </div>

  <div className="pt-7">
    <FilterButton />
  </div>

  <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-6 pt-6">
    {org.listings.map((item, index) => {
      <OrgFoodCard
        key={index}
        food={item}
      />
    }) }
  </div>
</div>
</div>
);
export default OrgProfile;
```

This is the code for the profile view for organizations. A lot of the view is similar to the user profile view, hence the use of helper components. But some features are different, most notably the star rating system. Also we show the items posted by the organization in their profile view.



Challenges and Insights

Challenges

- Code duplication: The views for organizations and users are similar but not exactly the same. And because we were working on these issues separately, we ended up with a lot of duplicated code. I ended up going back and doing a PR that created helper components. For example, both the user and organization profiles include a div with the user's location. Initially this was two separate divs, but I made one component with the location.
- Styling inconsistencies: Because we made the components separately at first, we ended up with inconsistencies. While the location div is now the same between user and organization, that was not always the case.
- Performance: We did not consider performance until the end when working on the checklist. While we followed some React best practices naturally, I ended up having to add React Router code splitting and lazy image loading after the fact.

Insights

- Plan in advance
 - People working on similar features should brainstorm which components will be shared and then make those first. They can then split up and make the overall feature using the helpers.
 - We should make a checklist of requirements in advance. For example, this time we should have said that we wanted to use lazy image loading.

Future Improvements

- We need to complete the ability for organizations to edit and remove listings. Once we have set up our backend, we can connect the form to it.
- Right now we have a toggle to switch between the user and organization views. But we should create a profiling system where each account is assigned to one of the two roles. Accounts will then see the view that corresponds to them. There can also be a public view that allows browsing without reservations.
- Our food cards are all shown on one page, but in production we would expect a lot of listings all at once, so I would split the home page into multiple pages. Perhaps each page shows 10-20 food cards based on the sort.
- We currently show the organizations' posts in their profile view, but I think we should consider making a separate view for that. It is possible that an organization has a lot of posts and the profile page becomes crowded as a result.