**Key purposes/social need:**
Neighborhood Connect is designed to allow the people of Boston to voice any service requests they have for any issues in their neighborhood, track the status of the request, and view any issues their neighbors have brought to light. The current system allows for the requests to be made but after that there is no visibility of the status of it. Our app will have a public forum that shows all requests made to allow others to upvote and comment on current issues and allow city officials to be able to get more information and communicate if an issue is currently being resolved or not. Neighborhood Connect will improve the lives of both city officials and members of the community by allowing for easy communication of collective needs and will make civil engagement simpler to use.

**Interesting and Substantive Conceptual Design Work:**
A user will be able to dynamically interact with reports and map visualization. A user will be able to comment, upvote, and post. A user will be able to filter by neighborhood. Admin will be able to mark issues with a priority level or as resolved. In normal reporting, a user is limited to creating and viewing reports. In our app, users will be able to comment on and upvote other users' reports. The App will also be more transparent than other interpretations, citizens can have access to reports as well as a map to visualize the reports. We imagine connecting the reports to a map will be challenging. While other aspects implement similar concepts to fritter, we have not yet incorporated a map visualization.

Key Concepts

1. **Name**: Account

   **Purpose**: store a user's information

   **State**:
   accounts: set Accounts, usernames: set String, passwords: set String
   accts: one usernames -> one Account
   auth: one usernames -> one passwords

   **Actions**:
   create(u: string, p: passwords)
   delete(u, auth)
   edit(u, auth, u')

   **Operational Principle**:
   u, u': User, p: Password
   create(u, p); edit(u, auth, u'); delete(u', auth)

2. **Name**: Post

   **Purpose**: publicly submit a service request

**State**:

Post -> User
Post -> set Comments, labels, votes

**Actions**:

create(p: Post, auth)
delete(p: Post, auth)

**Operational Principle:**

create() and not delete() => p in feed

3. **Name**: Comment

   **Purpose**: Discuss a post

   **State**:

   Comment -> one Post, User

   **Actions**:

   create(p: Post, t: text)
   C in P.comments
   delete(c: Comment)
   C not in P.comments
   edit(c: Comment, t')

   **Operational Principle:**

   create() and not delete()
   {C in P.comments}

4. **Name**: Upvote

   **Purpose**: track popularity of item

   **State**:

   Votes: Item -> set User
   Count: Item -> one Nat

   **Actions**:

   upvote(i: Item, u: User)
   i.votes += u
   unvote(i: Item, u: User)
   i.votes -= u if u in i.votes

   **Operational Principle**:

upvote()... unvote()
{i.count is #upvote() - #unvote()}

5. **Name**: Label

   **Purpose**: categorize an item

   **State**:
   Label -> one String
   Label -> set Items

   **Actions**:
   create(n: name)
   new Label(name)
   assign(L: label, i: Item)
   i in L.items
   remove(l: Label, i: item)
   i not in L.items
   delete(L: label)

   **Operational Principle**:
   After create(n) and assign(n, i) and not remove()
   {i in L.items}

6. **Name**: Session

   **Purpose**: auth for extended interaction

   **State**:
   name, password: User -> String
   Sessions: Client -> set User

   **Actions**:
   register(n: String, p: String)
   login(n: String, p: String, c: Client)
   logout(c: Client)
   auth(c: Client)

   **Operational Principle**:
   register(n, p, u); login(n, p, c)
   auth(c,u')=> u'=u
   logout(c); auth(c', u) => c' != c

7. **Name**: Neighborhood

   **Purpose**: group for users to participate in by viewing the posts of group members

   **State**:

   Neighborhood -> set Users
   Neighborhood -> one String Name, Description

   **Actions**:

   create(name: String, description: String)
   join(n: Neighborhood, u: User)
   u in N.users
   leave(n: Neighborhood, u: User)
   u not in N.users

   **Operational Principle**:

   after User u = create() and join() and not leave()
   {u in N.users}