6.1040 · software studio · fall 2023

design reviews

Daniel Jackson & Arvind Satyanarayan

purposes of today's class

inspire you by showing and appreciating great work

experience with you how to **critique** a design

give you **ideas** you can exploit in your final projects

point out pitfalls and common design **mistakes**

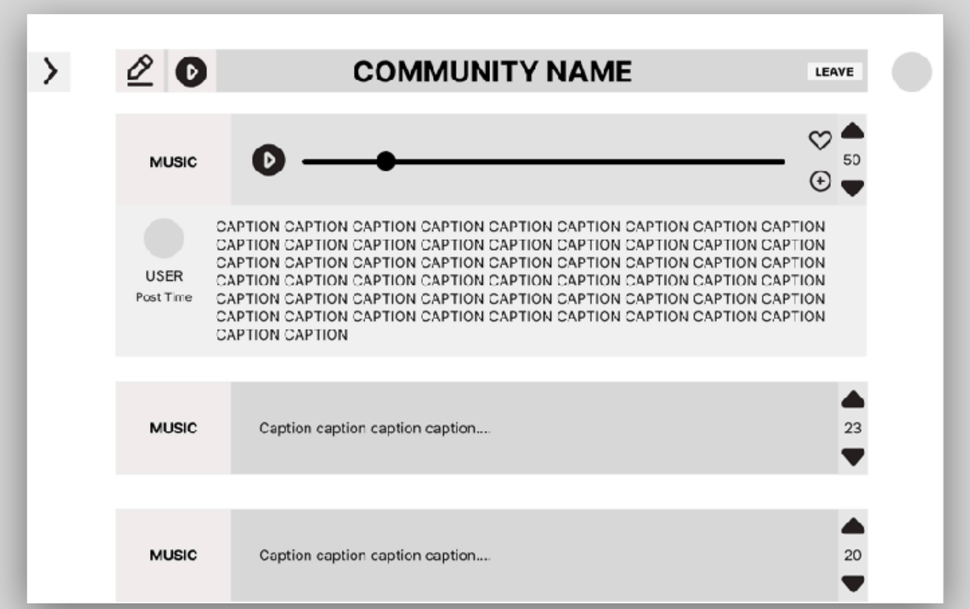
about the projects we're showing

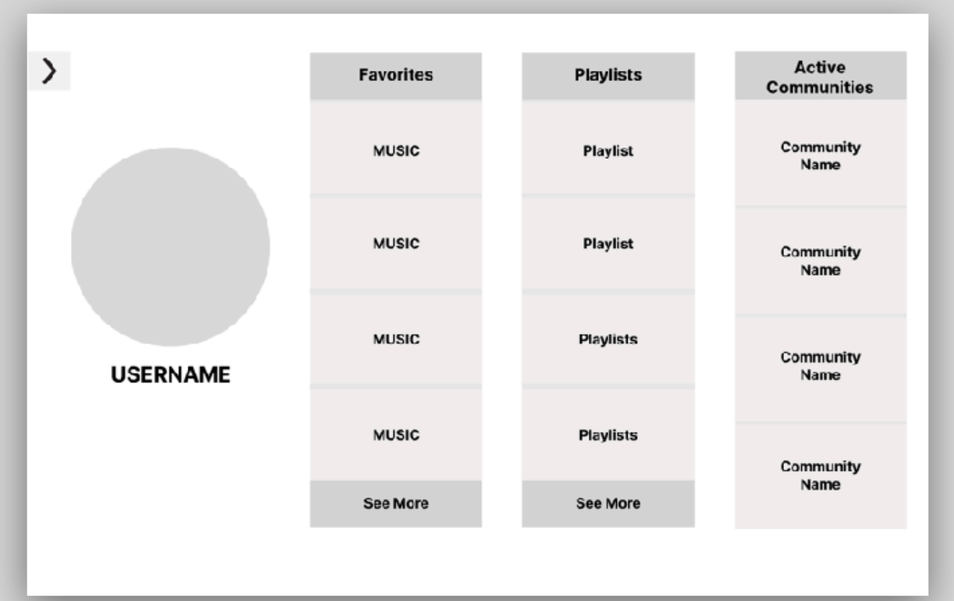
all are examples of **excellent work**

like all designs, **none are perfect**

just a sample, not the only great projects

sonvitas
(luca musk)





social communities built around musical genres
can post song with caption to community
automatically populates community playlist

variety of playlists
recent playlist, ranked by upvotes
official playlist, when members pin a song
personal playlists and favorites

clever concept design ideas

Compilation[Owner, Content]
Purpose

Manage content through compiling and ordering.

```
Include Compilation[User, Music]
Include Compilation[Group, Music]
Include Compilation[Group, Post]
```

Compilation concept
generalized over playlists & group messages
holds owner, keeps Group concept familiar
(no need to add playlists to Group)

Caption[Media]
Purpose

Enable users to add text to media

Caption concept
creates composite objects
for instantiation of group posts

```
sync upvote(voter: User, p: Post, l: Group[User]):
    assert Group.inCommunity(l, voter), p in Post.getPosts(l)
    when Vote.upVoteContent(p)
    Compilation.reorderCompilation(g, "recentsPlaylist", media ->
```

sync with Vote concept
used to curate playlist for group

concept design issues

Post[User, Content, Location]

Purpose

Publish content visible to other users

putting location in Post

factors sets of posts out of compilation and group
but playlists and chats have domain-specific properties
in particular, group has rules about who can post
result is more complex synchronizations
also Group OP is weak

a better approach?

remove Post concept
include posts in Group along with access rules
rename Compilation to Playlist
include songs in Playlist with cursor etc

```
sync post(u: User, m: Music, s: String, g: Group[User]):
    c = Caption.createCaption(m,s)
    assert Group.inCommunity(g, u)
    when p = Post.post(c, g)
    DatedObject.addItem(p)
    Compilation.addContent(g, "timeline", p)
    Compilation.addContent(g, "recentsPlaylist", m)
```

Group[Users]

Purpose: Group users together

Principle: An user can register for a group, leave a group, and be checked for member
in a group. New groups can also be created.

challenges & opportunities

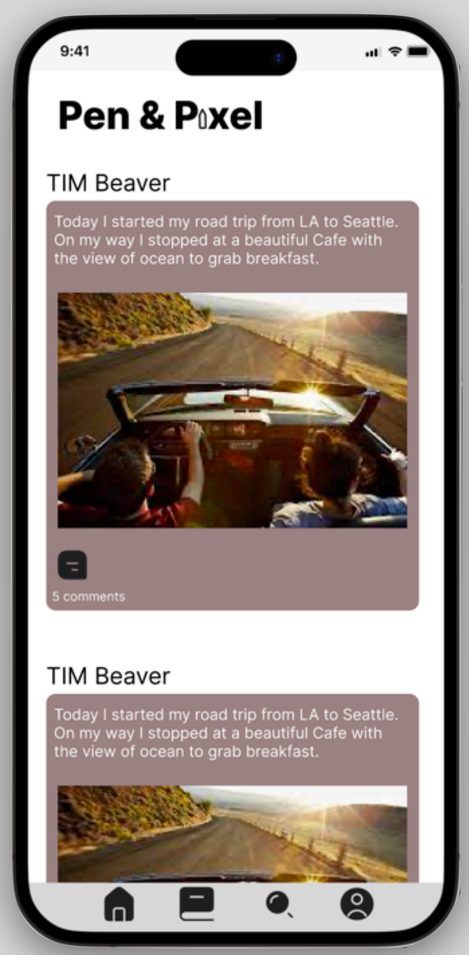
how to curate a group's official playlist?
let group have moderators and allow them to do it?

proliferating groups
are groups genres or friend groups?
how are they named and how do you find ones you like?

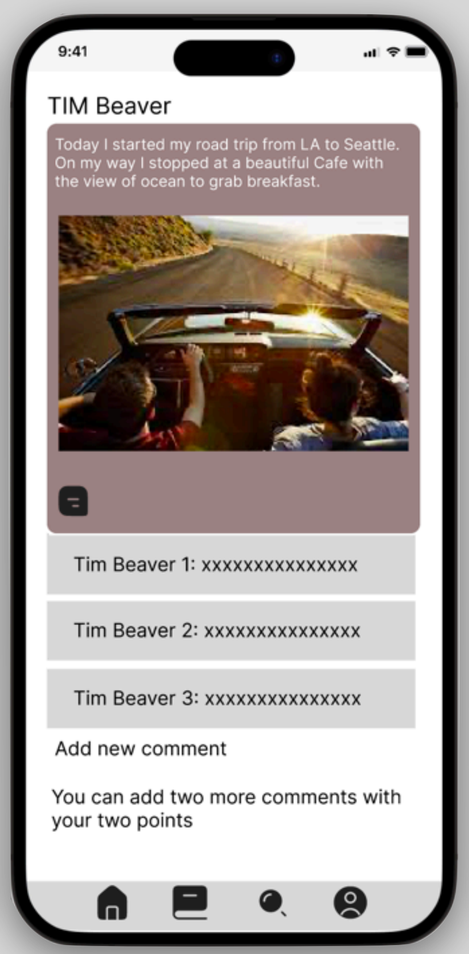
user feed
is there a feed of music posts independent of groups?
a way for users to discover groups?
how would it be filtered?

pen & pixel
(amirabbas kazeminia)

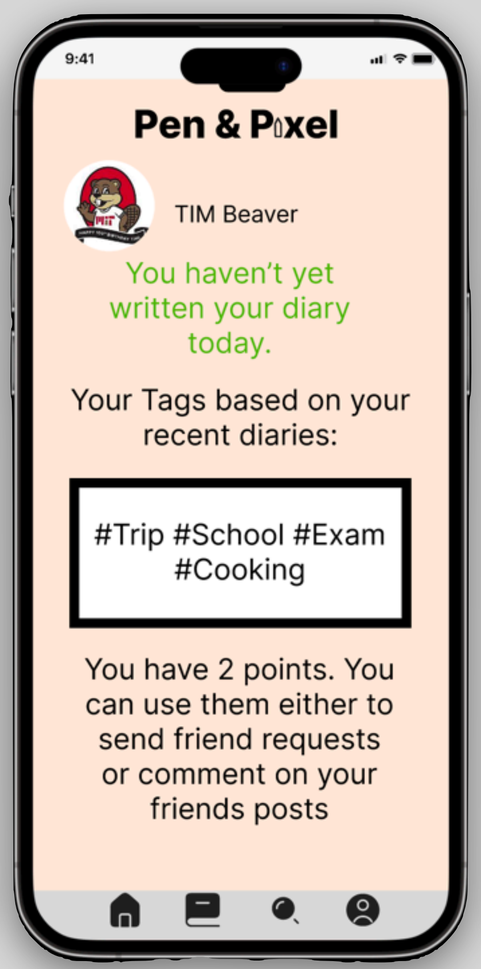
daily journalling



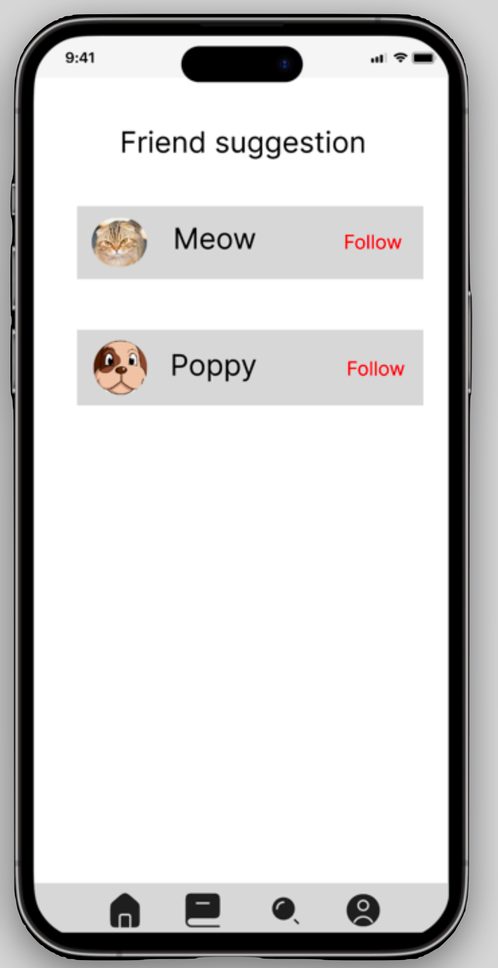
share images & text



others comment



points (karma)



suggestions based on content inferred by LLM

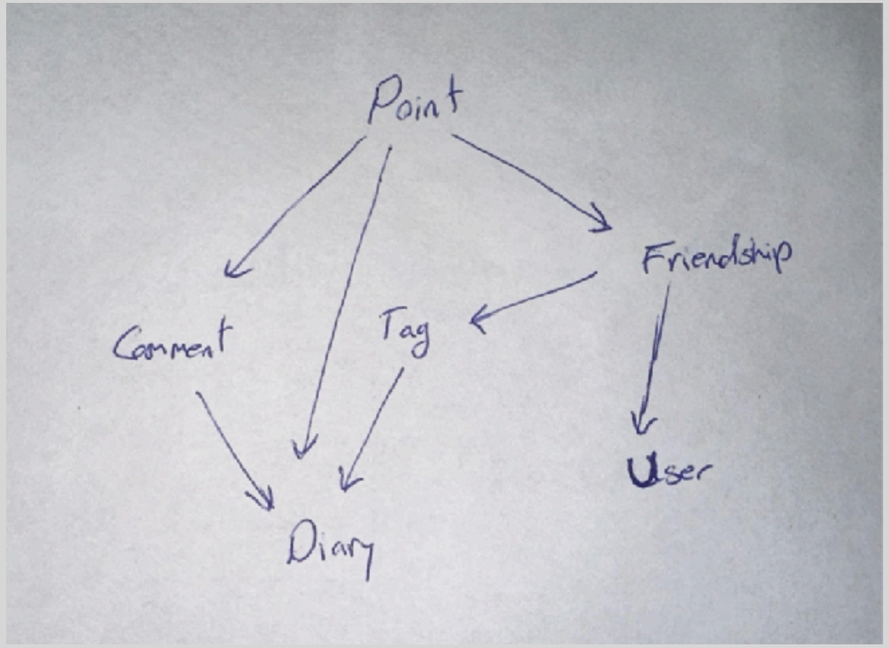
clever concept design ideas

Tag concept
user diaries used with GPT to assign tags to users
tags used to suggest friends

incentivization & control
karma points prevent friending until share diary
maximum of one diary post/day

separation of diary from posting
can write diary entry and not share
(not yet implemented, but easy)

Point concept
considered storing points in User concept
tried to implement and it became a mess
useful lesson about concepts vs OOP!



very focused design with few concepts
but Tag should depend on Friendship?

concept design issues

Concept	Tag
purpose	To label and categorize a person's journals
principle	It generates tags for a given text and keeps track of tags for a user
states	Tags: user → set String;

Concept	Friendship [User, Tag]
purpose	To connect users
principle	A user can send a friend request to any other users with one line message on why they want to be friends. The other user has the option to accept or decline the request. Once two users are friend, they will see their public posts. users that have similar tags

OPs of Tag and Friendship are weak
not clear in Tag what the tags are for
not clear in Friendship how tags added

- concepts
  - comment.ts
  - errors.ts
  - friend.ts
  - friendSuggestion.ts
  - point.ts
  - post.ts
  - tag.ts
  - user.ts
  - websession.ts

lack of modularity bites in code
added FriendSuggestion concept
but sync shows not encapsulated

```
@Router.patch("/friendSug")
async generateFriendSug(session: WebSessionDoc) {
  const user = WebSession.getUser(session);
  const userTags = (await Tag.getByUser(user)).userTags;
  const otherTagsDoc = await Tag.getOtherTags(user);
  const otherTags = new Map();
  for (const TagDoc of otherTagsDoc) {
    otherTags.set((await User.getUserById(TagDoc.user)).username, TagDoc.userTags);
  }
  const suggestion = await FriendSug.generateFriendSug(user, userTags, otherTags);
  return suggestion;
}
```

a more modular design

concept Friend
classic concept, no tags
includes published posts
OP: if you friend someone and they post, you can read the post

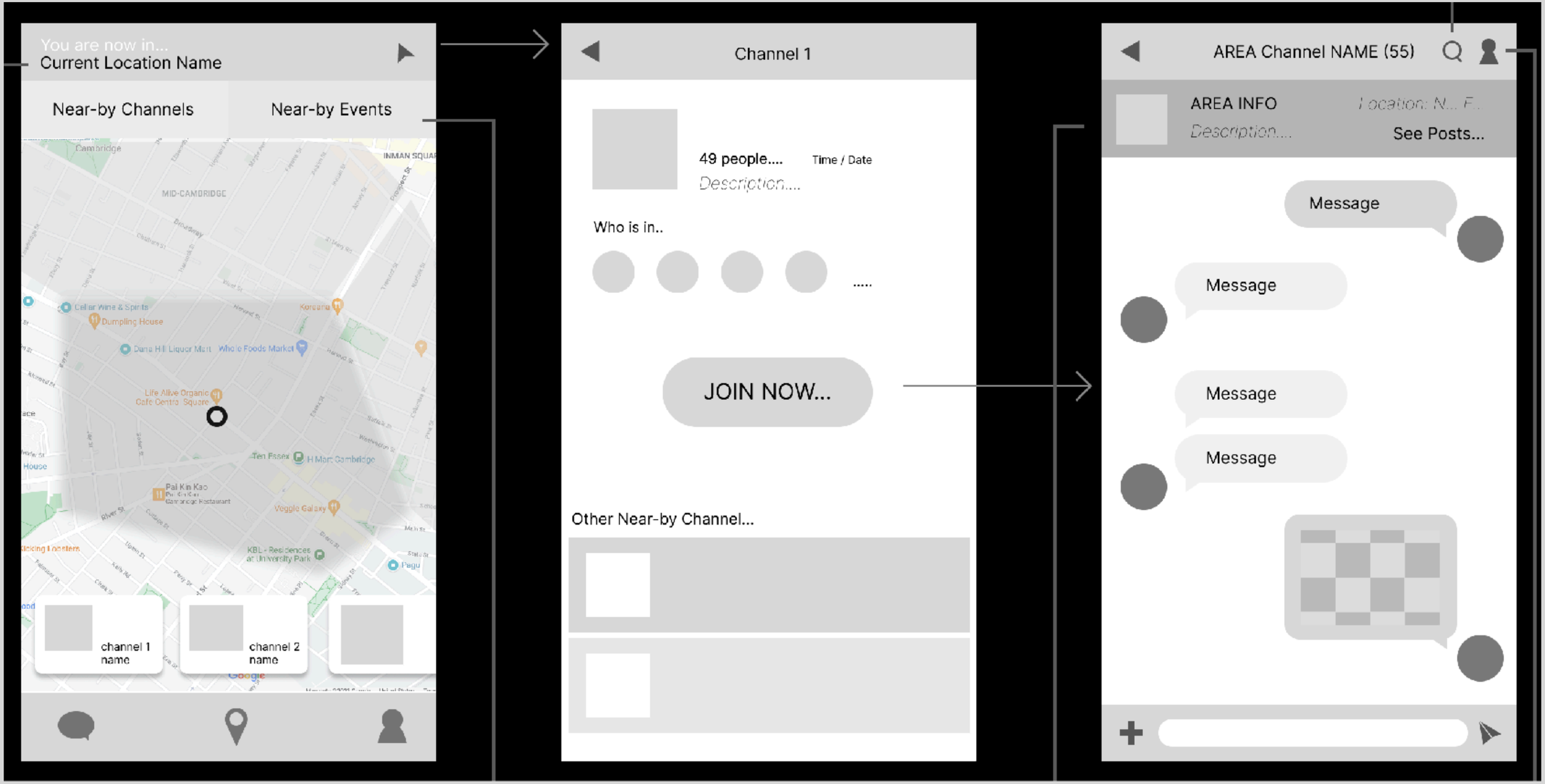
concept UserTag
novel concept
assigns tags to users based on text they write
suggests user connections, for friends and other
OP: add texts associated with users, then ask for suggested friends

challenges & opportunities

how are tags selected?
ie, about design of UserTag concept
currently fixed set of tags
could GPT choose the tags? what if they change?
should users see the tags?

how else could tagging be used?
ie, how to sync UserTag concept with others

localink
(yinghou wang)



events with locations have chat channels; users can join the channels if their physical location is nearby

clever concept design ideas

Channel

```
concept Channel [User, Locate]
    purpose
        authenticate the user to temporarily join the channel
    principle
        when the users u have the same location with the channel
        they are allowed to join the channel
    state
        channels: set String -> one Locate
        member: channels -> set User
    actions
        search-nearby (u: User, c: Channel, out bool)
        join (u: User, c: Channel, out mem: member)
        quiet (u: User, c: Channel, out mem: member)
```

Channel concept
Channel is an authentication concept
user can only join channel when physically close

LocateTag concept
assigns location tags to users, events & channels

```
sync register(u, p: String, out user:User)
sync login(u, p: String, out u:User, out s: Session)
        Session. start (user, session)
        ExpiringResource.allocate (session, 300)

sync logout (s: Session)
        Session.end(session)
        ExpiringResource.deallocate (session)
```

ExpiringResource concept
used to expire session (but not channel access)

concept design issues

Message

```
concept Message [User, Item]
    purpose exchange of message
    principle
        after a user sends a message to other users,
        they can receive that message
    state
        from, to : Msg -> set User
        body: Msg -> one Item
    actions
        send (from, to: User, body: Item, out m: Msg)
        recv (u: User, m: Msg)
```

Message concept

nice attempt to generalize over channel vs direct msg
but not clear how messages are stored in channel

```
// concept Tag [Item (a generic type)]
//      purpose
//              organize and show the new created item to users
//      principle
//              when a new item of content is created, it should be
//              categorized and taged with a set of labels(strings)
//      state
//              label: Item -> set String
```

new Tag concept added in code

so messages can be tagged with channels, eg

```
@Router.post("/message")
async sendMessage(session: WebSessionDoc, chatchannel: string, contents: string) {
  const sender = WebSession.getUser(session);
  const the_channel = await Channel.getChannelByName(chatchannel);
  const receivers = the_channel.members;
  const the_message = [];
  // check if the sender is in the channel
  if (await Channel.checkauthorized(the_channel._id, sender)) {
    for (const users of receivers) {
      the_message.push(await Message.send(sender, users, contents));
    }
  } else {
    throw new Error("You are not authorized to send a message!");
  }
  // assign each message with a tag of it's channel name
  const the_tag = await Tag.searchTag(chatchannel, "private_label");
  if (the_tag === null) {
    // should be a private tag
    throw new Error("Should be a private event for this channel");
  } else {
    for (const message of the_message) {
      if (message.message && the_tag._id) {
        await Tag.addItemtoTag(message.message._id, the_tag._id);
      }
    }
  }
  return the_message;
}
```

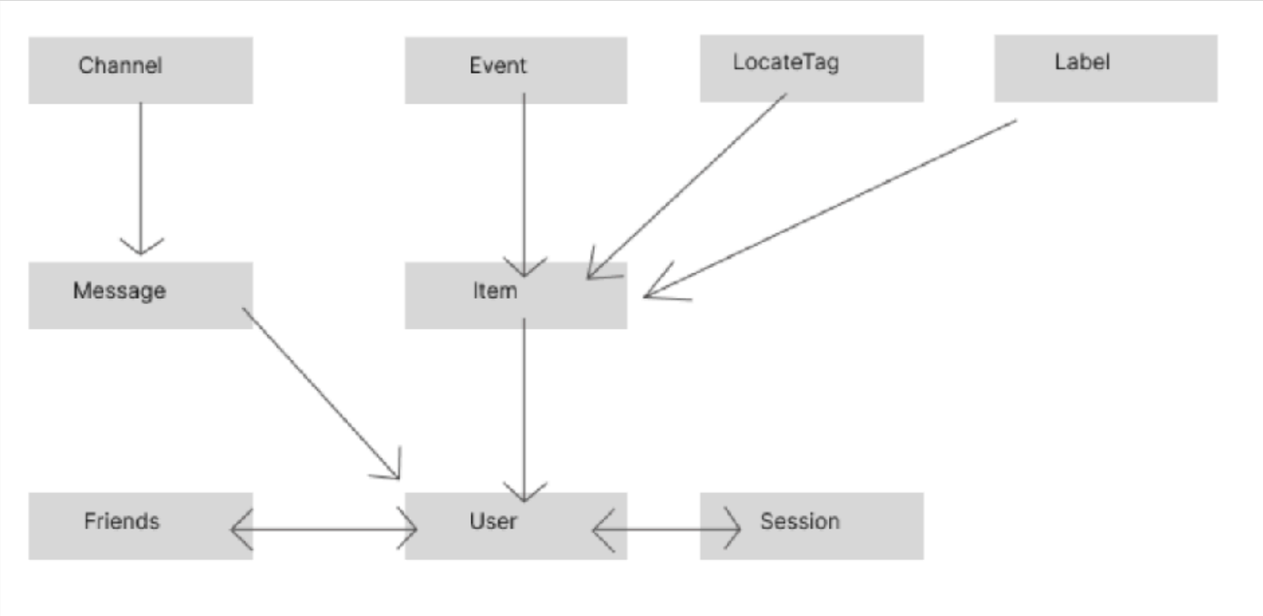
leads to complicated sync & lack of encapsulation

a more modular design

concept Channel
classic concept, just a chatroom
contains messages from members
and controls access to members

concept LocalResource
novel concept that guards access by location
can be applied by sync to channels and other things
encapsulates calculation of proximity
also handles timing: can you still access if in proximity time T ago?
maybe set expirations separately by channel

simplifications



remove some concepts
remove Event? just let Channel have header/profile
remove Friend? dependencies are unclear
remove Label? not clear what role it plays

challenges & opportunities

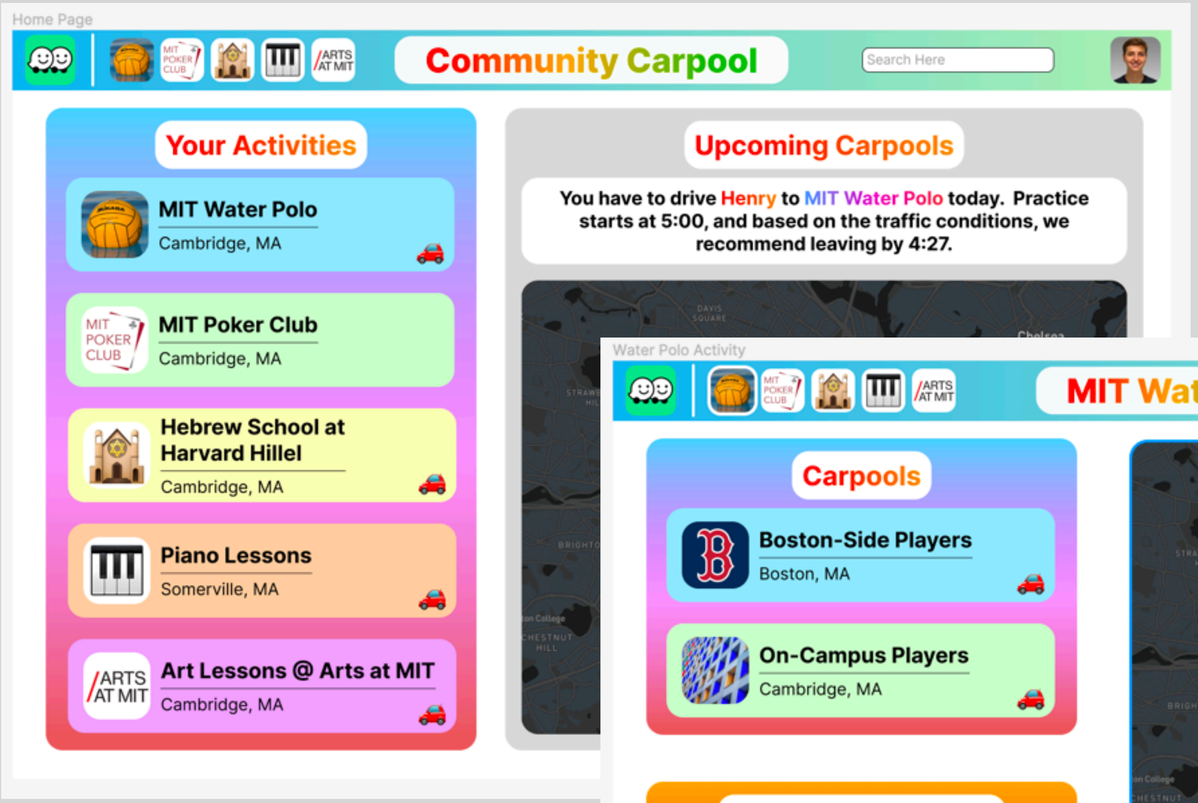
how is location determined?
in a museum or classroom, gain access via wifi?

what are the location rules?
was there in last hour? last day?

which actions are location limited?
joining a channel? posting in a channel?

precedents
FourSquare

community carpool (henry asa)



user has activities

each activity has carpools & activity chat room

carpool has chatroom & route

highlights

`Activity [User]`

Purpose

A subset of `Users` in a private group, where data is only accessible to those in the group.

Principle

`Activities` are subsets of `Users` who all share a commonality (participating in the same activity). Private messaging, information, and data can be accessed by `Users` who have been approved to join the `Activity` and is only accessible to `Activity` members.

Activity concept
joining activity is protected by password shared OOB

Publicity of `User's` `Location` Data

As part of the Community Carpool carpool coordination process `Users'` home addresses are viewable to all members of an `Activity` that a `User` is a member of. Assuming that people trust the other `Users` that they participate in these `Activities` with, this doesn't pose an inherent danger to a person (which is why I designed the application this way), but it could still be considered a little bit of a breach of privacy, as this rather sensitive location data can be viewed by others. The rationale was that this is not too different from a Phone Book, but I realize that these databases are not great for `User` privacy either.

Options

Define a Home Area Rather Than Address | Display Users' Addresses to the Activity

Rather than displaying `Users'` actual addresses to all of the *Members* of an `Activity`, a small radial circle that encloses a `User's` address could have been used, and then when a `Carpool` is configured, the actual address of the person would be shared with the members of that `Carpool`.

While this is a viable option that does not reduce too much functionality, it makes coordinating carpools a bit less convenient, as people would not see exactly where another `User` may live.

design tradeoffs well expressed & organized
eg, who is user location shared with?

concept design issues

`Post [User]`

Purpose

Enables `Users` to share information with each other.

Principle

`Users` can create content and upload it to Carpool Community as a `Post`, allowing other `Users` to interact with this content, `Comment` on it, and react to it. `Posts` are a semi-public form of communication, where `Users` choose which subset of `Users` can view the `Post`.

Post concept
has complex and unclear OP
seems to be coupled to Carpool
posts not yet carpool specific in code?

`User`

Purpose

Allow users to create a public-facing user profile so that they can use tha app.

Principle

Users are represented by user profiles, which enables users to register for `Activity Groups`, set their addresses, and serves as a one-stop-shop for information about that user.

`State`

```typescript
registeredUsers: set User
username: string, password: string -> register(u: string, p: string) -> User

name: string
address: string -> setLocation(address) -> Location
```

User concept
combines authentication with locations
beware of OOP temptations!

consequences of spreading functionality across concepts

```
export interface UserDoc extends BaseDoc {
  username: string;
  password: string;
  // address?: LocationDoc;
  posts: Array<ObjectId>;
  joinedActivities: Array<ObjectId>;
  joinedCarpools: Array<ObjectId>;
}
```

User concept includes Activity state

```
@Router.patch("/activities/join/:name")
async joinActivity(session: WebSessionDoc, name: string, join_code: string) {
  const user = WebSession.getUser(session);
  const activity = await Activity.getActivityByName(name);
  const members = await Activity.addUserToActivity(activity._id, user, join_code);
  return {
    msg: `User has been successfully added to the activity '${name}'`,
    members: members,
  };
}
```

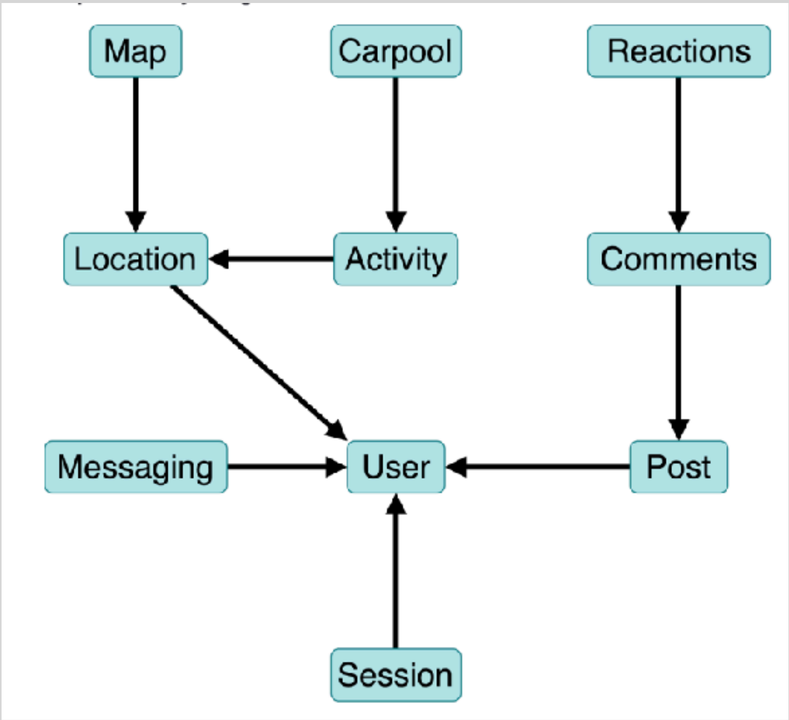
sync forgets to update User

a more modular design

concept Group
classic concept, just a chatroom
has profile/header, so can play role of Activity

concept Carpool
novel concept
has reference to a Group for conversation
and to associated activity (which is just another Group)
encapsulates functionality for planning routes
could also take candidate users and partition into carpools
note this is not a OO class!

simplifications



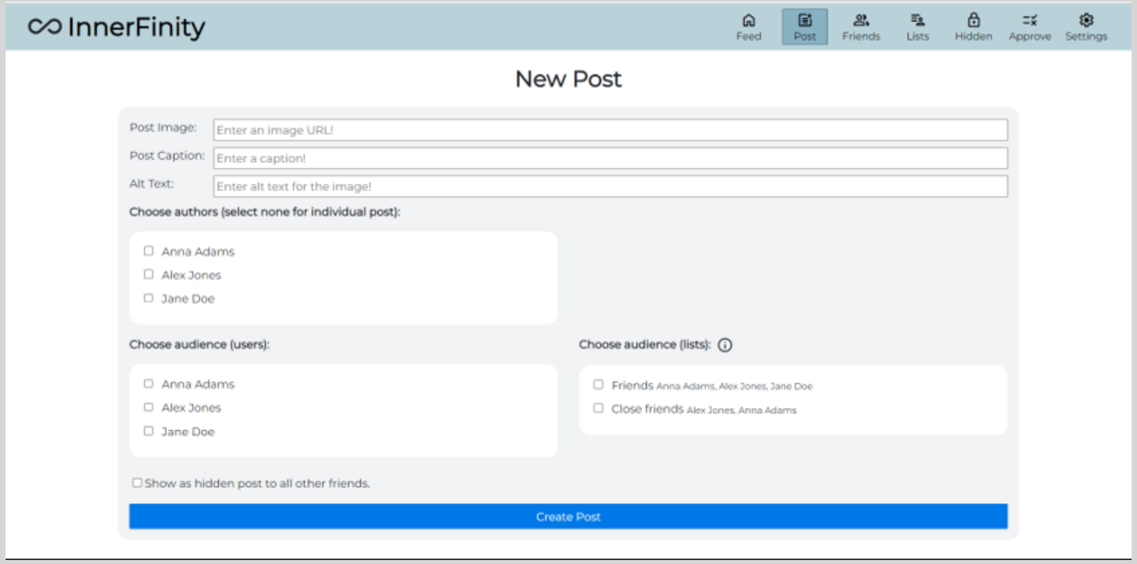
remove some concepts
Post enough: remove Comment and Reaction?
also remove direct Messaging?

challenges & opportunities

how are carpools constructed?
find route based on shortest detours
assign to carpools for best efficiency
rotating drivers (Henry discusses this)

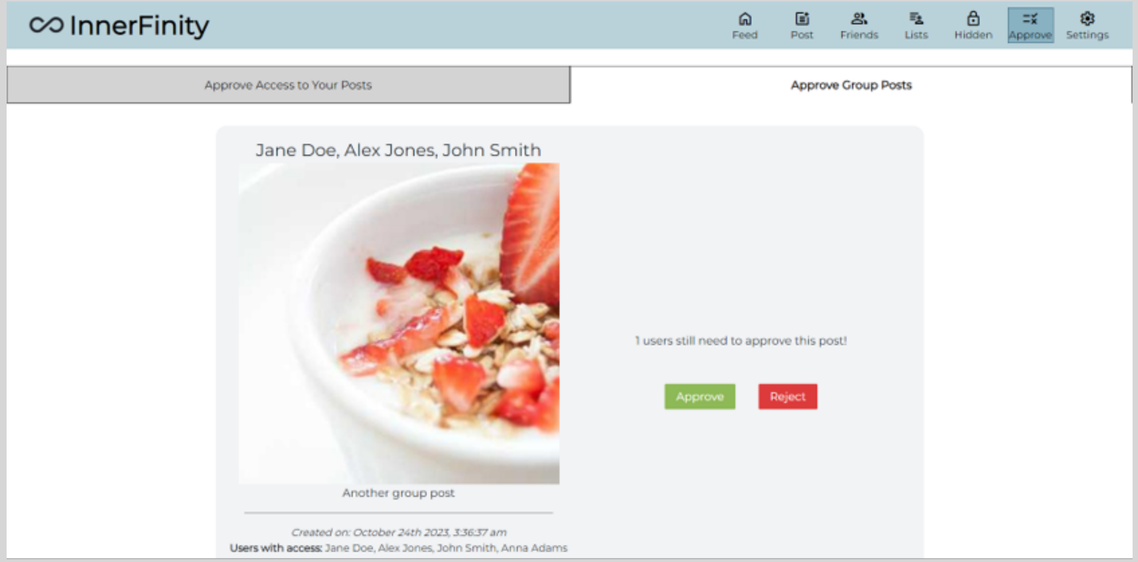
are carpools repeated?
one off vs regular?

innerfinity
(linda chen)



friends are divided into smaller “lists”

can grant access to lists or individual users



can name other users as joint authors
other authors must then approve

highlights

Concept Group[User]

- **Purpose** interact with users within a custom group
- **Principle** after a user creates a group, they can add members to or remove members from the group. All members of the group can see the group.
- **State**
  - groups: **set** Group
  - name: Group → **one** String
  - creator: Group → **one** User
  - members: Group → **set** User

Group concept (later renamed List)
generic over User, allows group to be a member

Concept Post[Authors]

- **Purpose** share content with others
- **Principle** after creating a post p, all authors must approve the post if there are multiple authors. After a post has been approved, a post p will be posted and other users will be able to see p.
- **State**
  - pendingPosts: **set** Post
  - publishedPosts: **set** Post
  - authors: Post → **one** Authors
  - content: Post → **one** Media (Note: Media is just an Image and String.)

Post concept
really nice, note strong OP (but needs more state)

Concept Sharing[Resource, User]

- **Purpose** control access permissions for a resource
- **Principle** the owner of a resource limits access permissions for a resource to a subset of users U, and chooses whether to allow other users to request access. After creating the Sharing permissions, all users in U can see the resource, and all users not in U can request access to the resource. After a user requests access, the owner approves or denies them access. The owner can also separately add or remove access for users.
- **State**
  - shared: **set** Sharing
  - owner: Sharing → **one** User
  - content: Sharing → **one** Resource
  - withAccess: Sharing → **set** User
  - allowRequests: Sharing → **one** Boolean
  - requestedAccess: Sharing → **set** User

Sharing concept
enables fine-grained ad hoc sharing

concept design issues

our forum discussion
initially wanted to post as group
a concept overloading: two purposes for Group
privacy issue: group names became public
encapsulation issue: approval becomes part of Group

```
@Router.get("/accessiblePosts")
async getAccessiblePosts(session: WebSessionDoc) {
  const user = WebSession.getUser(session);
  const userLists = (await UserList.getUserListsByMember(user)).map((x) => x._id);
  const resources = await PostSharing.getResourcesByAccessible(user, userLists);
  const postIDs = resources.map((record) => record.resource);
  const posts = await Post.getPublishedPosts({ _id: { $in: postIDs } });
  return Responses.posts(posts);
}
```

Sharing and List concepts are not cleanly separated
list members passed to sharing for check each time

a more modular design

concept AccessList [User, Item]

purpose manage access to items through flexible lists

principle user creates list and grants access to item for that list, then members of the list can access the item

state

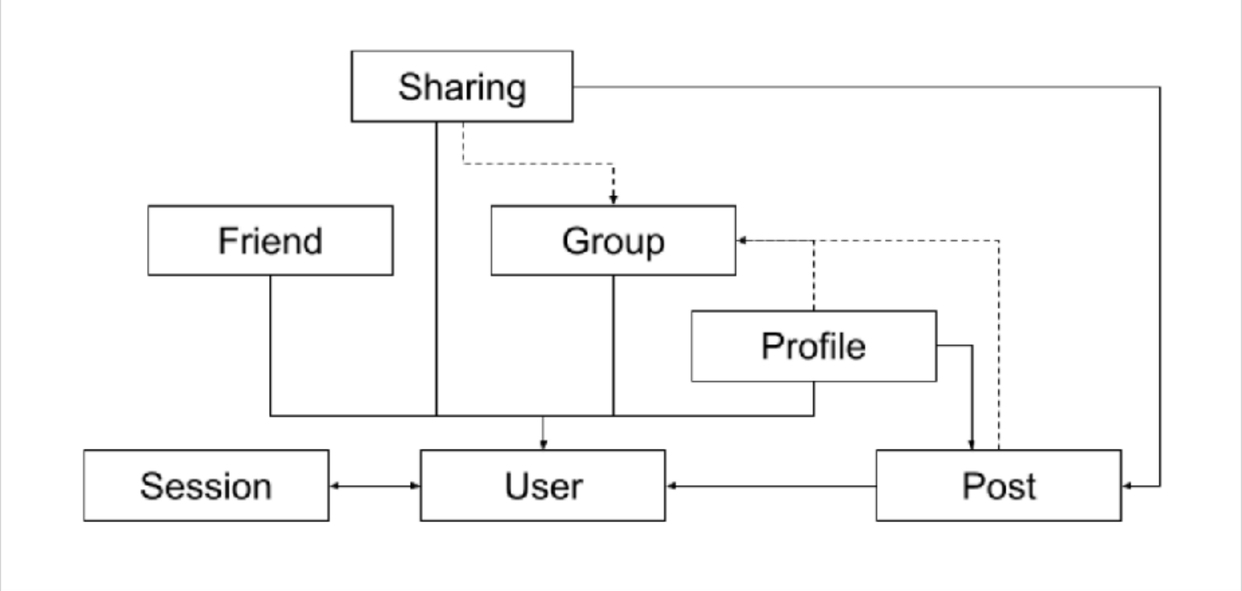
owner: List -> **one** User
members: List -> **set** User
name: List -> **one** String
author: Item -> **one** User
access: Item -> **set** List + User

actions

create (owner: User, name: String, **out** list: List)
add (u: User, l: List)
grant (i: Item, x: List + User)
access (u: User, i: Item)

encapsulates management of lists and granting permissions

simplifications



can Friend be removed?

already treated in UI as a named list but has its own concept

allow some lists to be public and have request to join?

challenges & opportunities

Google +
(2011-2019)

is management of lists too much work?
maybe shared lists (eg, WhatsApp) easier?

collaborative authoring
could this be the sole basis for a new app?

takeaways

concept >> class
a concept can be more than a class, multiple collections
eg, Carpool has users to match as well as carpools
eg, Friend has requests and accepted

encapsulate state/actions by function
don't allocate state based on where you'd put an attribute
avoid coupling between concepts
eg, user location belongs to location-specific concept, not User

concepts embody rich behavior
enough state and actions to embody rich behavior in concept
tipoff: weak operational principle, syncs with control flow or maps
eg, separating Msg from Group prevents Group controlling access

simplifying
most of these have one really novel idea
can trim away some of the other concepts

example of a clean sync

```
@Router.delete("/posts/:_id")
async deletePost(session: WebSessionDoc, _id: ObjectId) {
  const user = WebSession.getUser(session);
  await Post.isAuthor(user, _id);
  await Comment.deleteByTarget(_id);
  return Post.delete(_id);
}
```

from Amir's code

successful app = novel concept or novel sync

location-based access, collaborative post

playlist x group, channel x location, chat x carpool

biggest issue = lack of encapsulation

seen in weak OPs refs to other concepts

results in complex syncs and loss of modularity