PlanIt Design Plans and Models

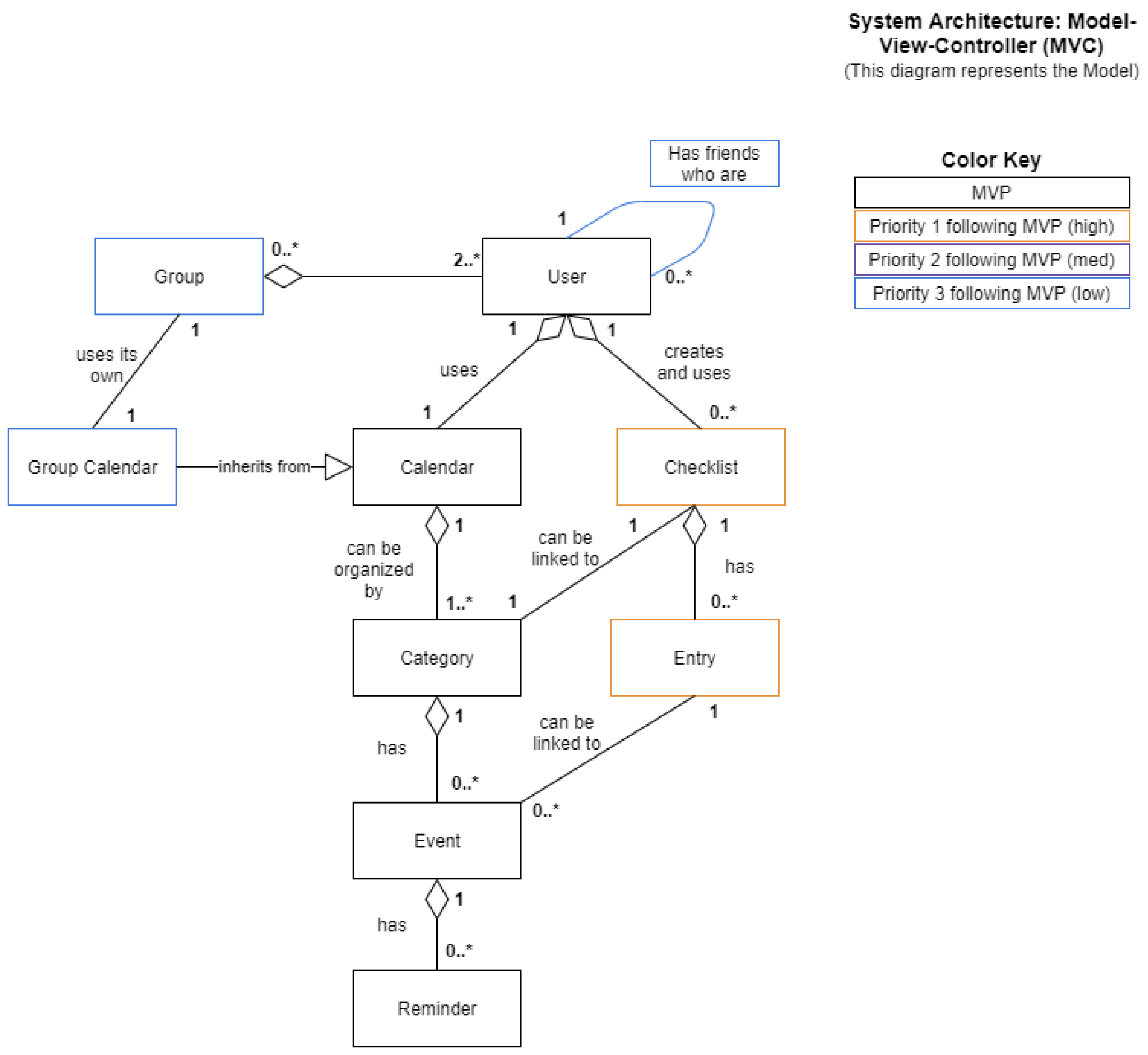
Design

Design Patterns / Architecture:

The architecture of our calendar is going to follow the Model-View-Controller (MVC) convention. Inside the Model, we plan to use an Object-Oriented approach, to capitalize on any potential abstractions there may be. For example, our Group class will have its own Group Calendar, which will inherit from a regular Calendar. We recognize that most of our MVP wouldn't technically need an O-O approach; however, building up with an O-O approach will help with expandability and maintainability.

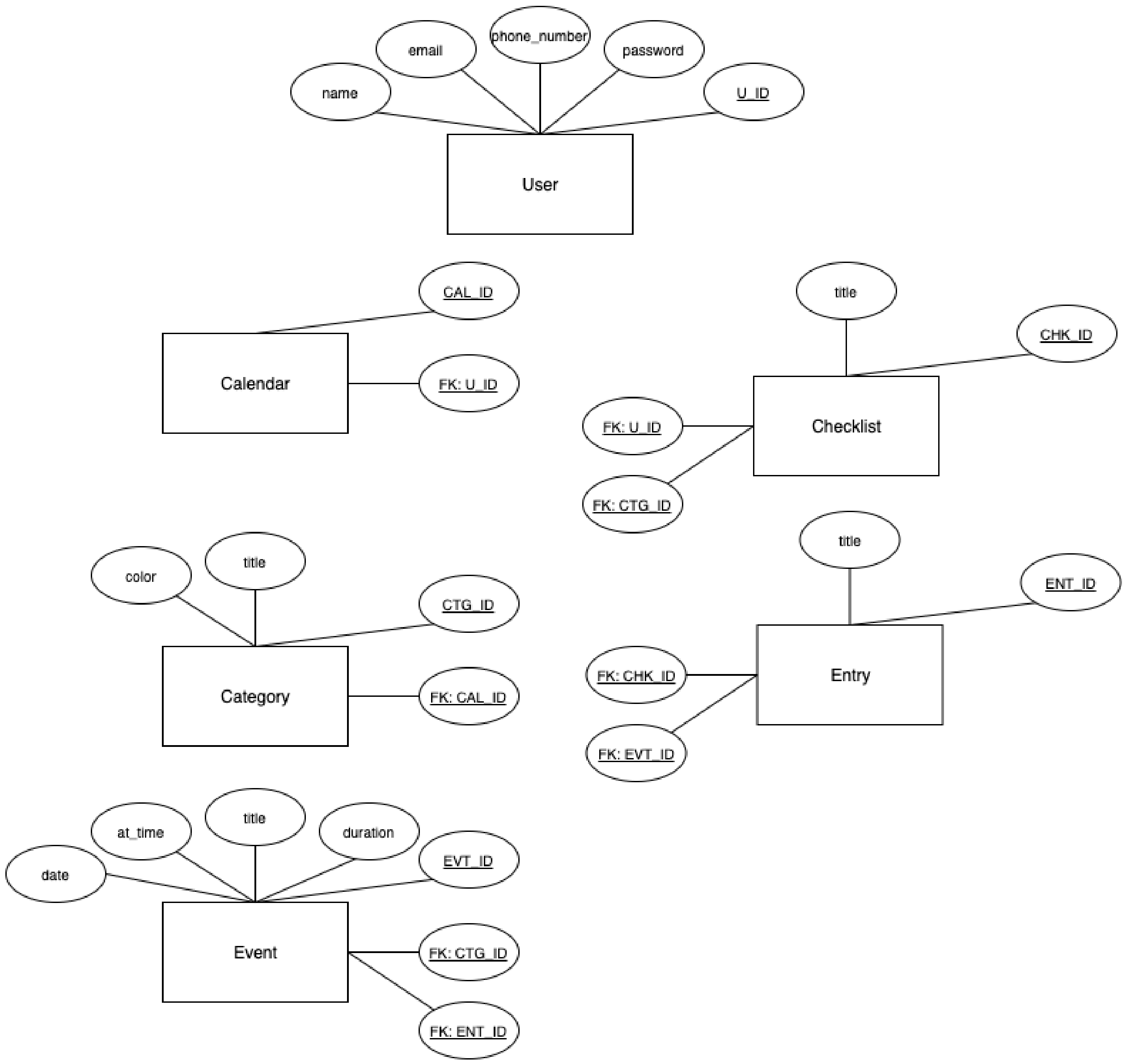
We chose to use this design model because MVC supports a faster development process through allowing parallel development. With the little time we have to develop our calendar, this is absolutely imperative for us. The MVC architecture is also commonly used for websites, so there might be helpful resources already out there to accommodate this method of organization. The MVC will help guide us a lot as we have little experience putting separate pieces of a project together. A principal idea of the MVC is that a programmer can make modifications to their part without affecting the entire program. With five people, it is important that we don't modify other team members' code unintentionally, or have any redundancies.

Class Diagram:



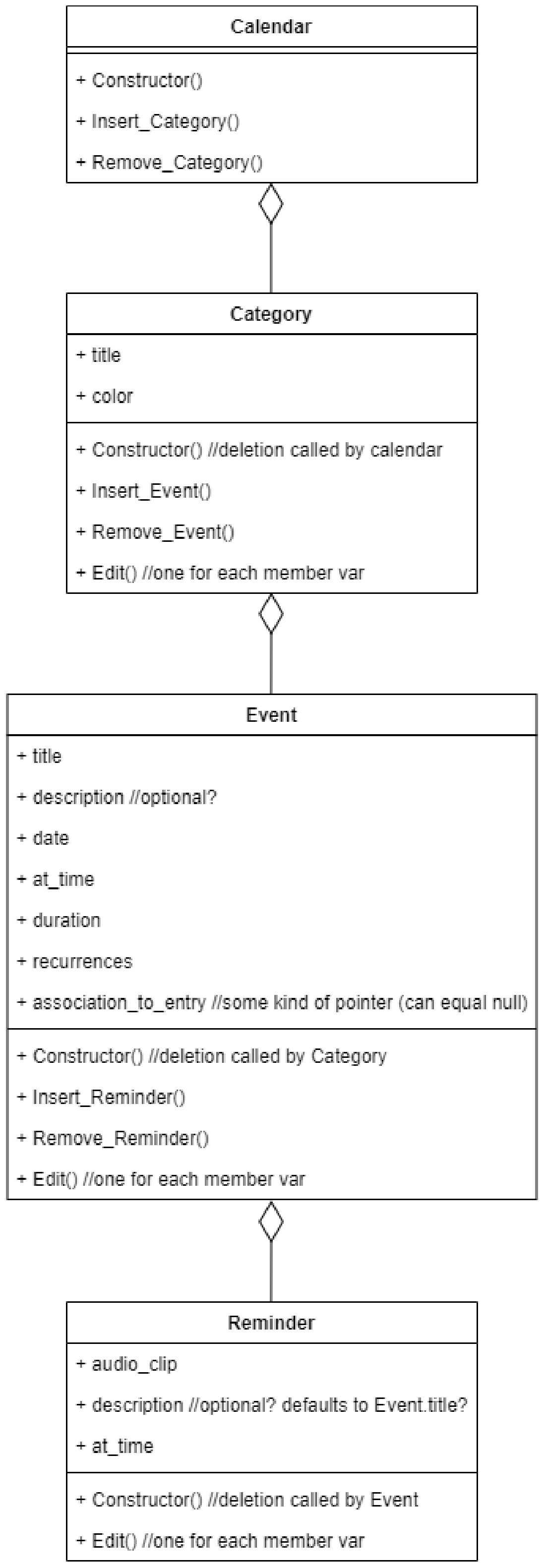
A quick note: Not included here is the Chat object. This would theoretically have a relationship between Users who are friends, as well as a relationship with Group. It's simply low enough in priority that it's not worth considering in finer detail at this time.

Entity-Relationship Diagram:



Our entity-relationship diagram does not include any diamond-shaped relationship tables. This is because all of our relationships can be linked with a one-to-one or one-to-many relationship, and so all foreign keys can be included inside the tables of the classes themselves. For example, Category contains a FK:CAL_ID**,** which gives the relationship between a Calendar and the Calendar's Categories. Likewise, Events are linked to Categories. Notice that Events are also linked to Entries by way of the FK: ENT_ID. This is because we want to give users the option to form an association from an Event to any Entries in a Checklist (but this is optional, so note that the foreign key here can equal NULL). See the main Class Diagram above for another visualization of this relationship.

Calendar Object Diagram (detailed):

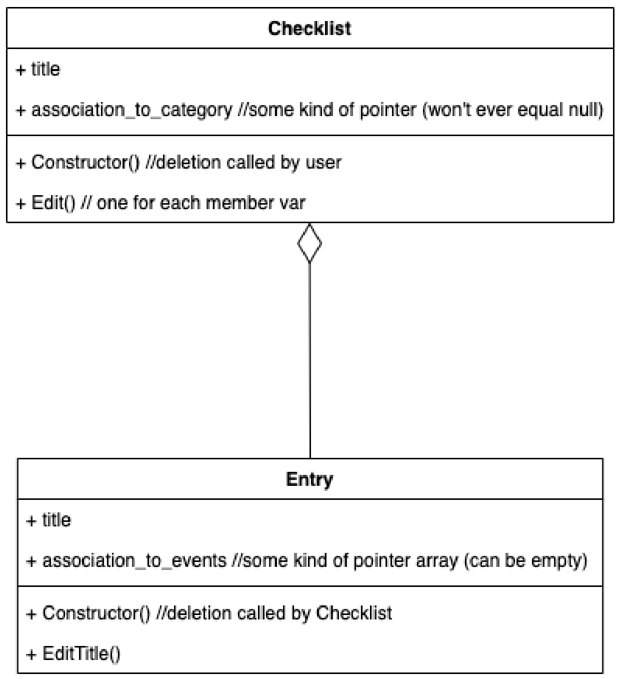

Calendar
+ Constructor()
+ Insert_Category()
+ Remove_Category()
Category
+ title
+ color
+ Constructor() //deletion called by calendar
+ Insert_Event()
+ Remove_Event()
+ Edit() //one for each member var
Event
+ title
+ description //optional?
+ date
+ at_time
+ duration
+ recurrences
+ association_to_entry //some kind of pointer (can equal null)
+ Constructor() //deletion called by Category
+ Insert_Reminder()
+ Remove_Reminder()
+ Edit() //one for each member var
Reminder
+ audio_clip
+ description //optional? defaults to Event.title?
+ at_time
+ Constructor() //deletion called by Event
+ Edit() //one for each member var

Purpose of association_to_entry:
1. This will hold a pointer if a User has spawned an Event off of an Entry. Entries and Events that are linked can be used to reference one another efficiently. Otherwise it will be null. For a full explanation of that functionality, see the Checklist Detailed Object Diagram.

2. Q: Why can it equal null? A: Users can create Events which aren't linked to any Entries in a Checklist. We would imagine this will be null more often than not, actually.

3. Q: Why doesn't Category have an association_to_checklist member variable? After all, Checklist has an association_to_category member. A: When creating a Checklist, some of its graphical elements and functionalities include having an associated Category (color-coordination, user organization, Event-spawning efficiency), whereas creating a Category doesn't actually ever need to access its relationship with any Checklists for functionality.

Checklist Object Diagram (detailed):


Checklist
+ title
+ association_to_category //some kind of pointer (won't ever equal null)
+ Constructor() //deletion called by user
+ Edit() // one for each member var
Entry
+ title
+ association_to_events //some kind of pointer array (can be empty)
+ Constructor() //deletion called by Checklist
+ EditTitle()

Purpose of association_to_category:
1. The user's organizational preferences: This association can be used to display Category's color theme in the Checklist View, as well as some verbal annotation that the Checklist is associated with this category. (e.g. Agenda : School -> Checklist.title : Category.title)

2. Entries: when a user creates an Entry in their checklist, they will have the opportunity to spawn specific Events that link back to the Entry. These Entry-spawned Events will pre-set the Category.

3. Q: Why is it never null? A: The running plan is that there will be an un-deletable Category titled "General" which be the default association, unless the user says otherwise.

Purpose of association_to_events:
1. As mentioned above, Entries will offer the User the opportunity to spawn Events which are linked to that Entry (this is a single-Entry--to--multiple-Event relationship). When any Event is spawned, its association is appended to Entry.association_to_events. Implied is that if an Entry-associated Event is deleted, its association must be deleted here.

2. If the user is in Checklist View, some indication will show them any associated Events next to their respective Entry. If this indication is clicked on by the User, the system moves to the Calendar View, and brings up the proper day in the calendar where the Event is set to occur.

3. Q: Why can it be empty? A: This is the case when the user has not spawned any associated Events with the Entry, which is fine.

Something to note here: We're still concerned how to represent Checklists on the View side. Basically we want Checklists to be blank documents that users can write into, with some added functionality (if we can get to it) to be able to spawn Events off of any Entry in a Checklist. We might need to look into some premade framework. We'll be thinking about it!