Software Design: Assignment 4

### Flow of content into the system from external sources

1. 1. Potential issues involve: Storing login credentials, Allowing 3rd party login, Enabling two factor, validating email addresses, role management
   2. The login/security considerations could be handled by implementing a UserAuth service provider that hashes and stores login credentials, offers two factor support, and can validate 3rd party accounts for creation. Validating email address uniqueness (or username if accounts are public facing) and user roles should be handled by the UserCollection when attempting to create and validate a new account.
2. 1. Having users randomly enter keywords would likely result in poor functionality, because there might be no corresponding tells.
   2. By implementing a TellSubscriptionController, you could have said controller provide a keyword set to the front end so users can select from a list of already associated keywords.
3. 1. Confirming file types, handle teller approval, respond to network failures, handle simultaneous uploading
   2. The TellHostInterfacer can handle file type confirmation and flag for teller approval, since it will be uploading the tell data. The actual managing of the live recording should take place in the controller, and it can direct the various TellHost inputs to appropriate actions, and detect network failures and direct to services to recover.
4. 1. Format, venues that don’t exist, consistency and error checking, dealing with conflicts
   2. The FestivalSchedule should implement an unmarshaling and serialization strategy to convert festival data for the database. That process should also include data validation, error checking, correcting naming inconsistencies, flagging schedule conflicts and passing on venues that don’t yet exist in the system to be created.

### 

### Flow of Tells from TellHostController to the TellEventController

1. 1. One pro is we do all our tell creating in batches, which adds additional security because we don’t accept random pushes. This could be useful if tells are uploaded sparaticlly, and their creation is not time sensitive.
   2. Offers real-time creation for tells, simplifies logic and doesn’t require polling.
   3. No real advantages, you are essentially coding two patterns that aren’t interacting. This could result in poor communication, duplicate message sending, and poor maintainability and testability.
2. We’re going the push route, as it seems to make the most sense to generate tells as they are finished being recorded.
3. See diagram
4. Yes, you can indicate forks and joins in an activity diagram. This allows us to model the concurrent flow of different threads, while indicating a merge point such that we can proceed properly after the multithreading execution has completed. For the case of uploading, this could be more complicated, because an activity diagram models one series of events, and you would likely need to spin up new threads on demand and before other threads complete their upload. So while an activity diagram could model a simplified example of this process, it would not fully represent the nuance of the pattern.
5. 1. Through VenueCollection
      1. I think this is a bad idea, as it involves tight coupling of a structurer to a process for a separate entity (the tell). I’m not sure what benefits this would provide
   2. Through TellHostInterfacer
      1. This makes much more sense, and isolates the Tell upload logic to the Tell Host entities. Keeping the TellHostInterfacer related to the venue collections is a good idea, as it lets us provide that information on upload. Maintainability and testability are the greatest pros for this approach.
6. Through the TellHostInterfacer, as it isolates logic well, making for a cleaner design.

### 

### Flow of Tells from TellEventController to the TellCollection

1. 1. The flow is quick and less complex by directly connecting components, reducing latency. This leads to tight coupling, making testing and maintaining more difficult.
   2. This option is less complex than a pull, but does a good job of isolating actions to their corresponding classes. Going through the collection as opposed to the info holder is the best route. The main benefit here is the collection is responsible for creating a new member, and the tell event controller is still only responsible for routing data to appropriate actions.This makes testing and maintaining much easier.
   3. The main benefit here is that tells are created in batches, allowing you to more easily isolate changes. The drawback to that approach, however, is if our cache fills up before we are polled, we could lose data.
   4. This is similar to the last example, in that its strengths are out tells are created in batches, but is a worse implementation, as it requires the TellEventController directly creating Tells to be later added to the TellCollection. This is a case of poor separation of concerns, in which many classes are extending past their intended functionality.
2. Option b, the sub option of going through the tell collection, because it allows us to generate tells as soon as they are completed, but avoid coupling issues present in a.

### 

### Flow of Tells from TellCollection to the NotificationService

* 1. This seems like a good option, as it separates the concerns better than the others. By having the notification service maintain responsibility for generating the notifications as opposed to the tell collection, this code is easier to test and maintain. Here, the tell collection only worries about maintaining its collection, and invokes the notification service to handle the notifications.
  2. Both versions of this approach seem problematic, because they both involve a tell collection writing an update notification. This is a bad example of coupling, in which a collection class is generating information holders unassociated with its collection. I don’t believe this approach offers great benefits, and should in face be avoided.
  3. The main benefit to this approach is if you wanted notifications to go out in batches. For instance, you might not want to alert people over and over again as tells are completed, since things are being recorded in real time. For example, you have several tell events going on at various venues all scheduled from 7-8pm. As they begin to finish, the tells are generated, so they come in at staggered times, some a bit before 8 and some after. As a result, if you are pushing notifications every time a matching tell is uploaded, a user with many subscriptions may receive a series of annoying notifications, one after another, as the events end in real time. If you, however, were to poll a bit after the hour, say 8:05, you could generate notifications for all the tell events that ended in the last hour, and push those out at once, so that same user would all their notifications at one time.
  4. This has the same main issue as option b, in which the tell collection is responsible for writing an update notification. I think this is a troublesome design and should be avoided for the same reasons.

1. Option b, because I think generating the notifications in batches makes the most sense considering the tell events will finish in real time and we don’t want to overwhelm our users with a staggered series of notifications.
2. See diagrams.