University of Regina

# CHECK THE LIST
# PROJECT UPDATE

Vaibhav Sharma, Melanie MacDonald

200365101, 200210467

CS455

Instructor: Dr. Orland Hoeber

Due: Nov 16, 2018

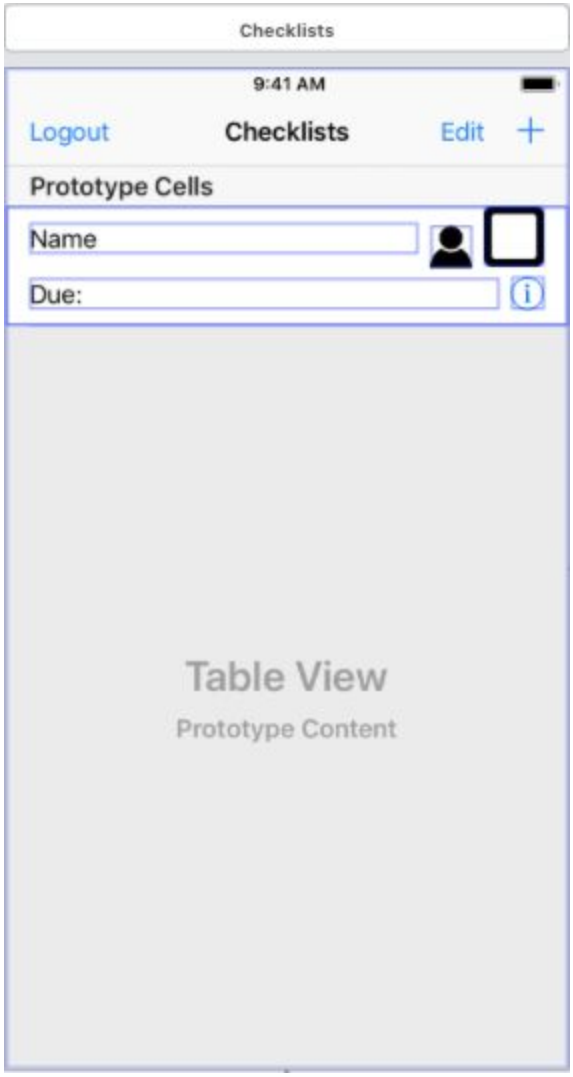
# Revised Project Proposal

This app is intended to store multiple checklists that a user can create and share with others, such that multiple users can collaborate on a checklist, and further on certain tasks which comprise a checklist. The main purpose of the app is to facilitate the organization of one or many people on a potentially shared project or endeavor by allowing them to represent their project as a checklist that is further broken down into tasks. The app provides status updates for all members that can be updated and viewed by anyone curious on the progress of tasks in the list. The information on these checklists are stored externally in a shared database, as each user is required to have the same information if they are collaborating on the same project. The technology we are using for this is Firebase, a cloud-hosted database that will asynchronously update the information provided in the app to facilitate coordination between users on a task. The idea is that if one person creates a checklist and invites others to it, from there all participants will gain access to that checklist after the app receives that information from Firebase.

The key features of this app are to create/modify/delete checklists and tasks, to include other users as participants on a checklist or task, to display the status of a task or checklist, and to synchronize the status or any changes to a checklist or task across all participants of that item. The main information provided by a checklist is the title, a description, who the participants are, when the checklist is due, and whether the checklist has been completed or not. The main information provided by a task are the same as a checklist in terms of title, description, due date, and participants, but in addition to indicating the completion status of a checklist, other status indicators are provided, the options being “Available”, “In progress”, or “Completed”.

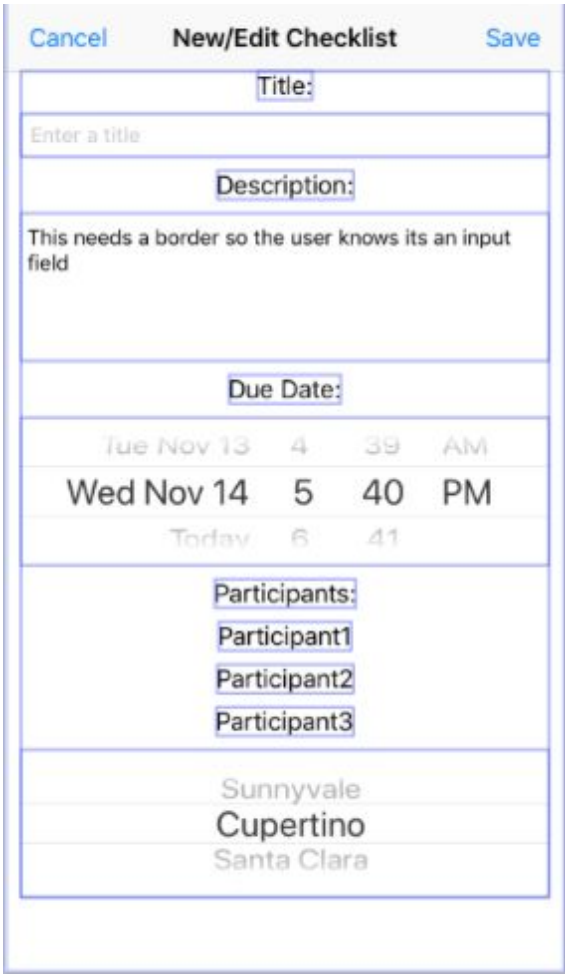
# Progress on Key Features (Frontend)

- List Checklists/Delete




This screenshot is the current prototype for the main view the user sees upon logging in, which lists all the checklists they are currently a part of. Each cell is laid out as follows, showing the most important pieces of information at a glance. The image resembling an empty checkbox on the far right starts out with that image on creation of the task, and changes to a checkbox that is "checked" upon task completion, so the user can quickly see whether the checklist has been completed or not. The image that looks like a person may or may not be hidden, depending on whether or not that checklist has been shared with any other participants. If the icon is absent, then there are no participants besides the active user. The icon that looks like an info button will reveal extra details about the task when clicked, those details being the description and all participant usernames. For navigation, the navigation bar includes a "Logout" button that will sign the user out and return to the login view. The "+" button will navigate to a view that facilitates checklist creation, and tapping on a tab row outside of an input field will navigate to a view that shows the checklist information for modification. The "Edit" button at the top facilitates deleting checklists by using the default functionality provided for removing rows in a table view. The view currently has proper auto-layout constraints to support either portrait or landscape orientation for any device. Currently the view only accepts serves to display what information will be displayed, with some navigation implemented, but the implementation for retrieving the checklists from the database, deleting the checklists, or modifying the cell appearance based on that information has not yet been implemented.

- Create/Modify Checklist

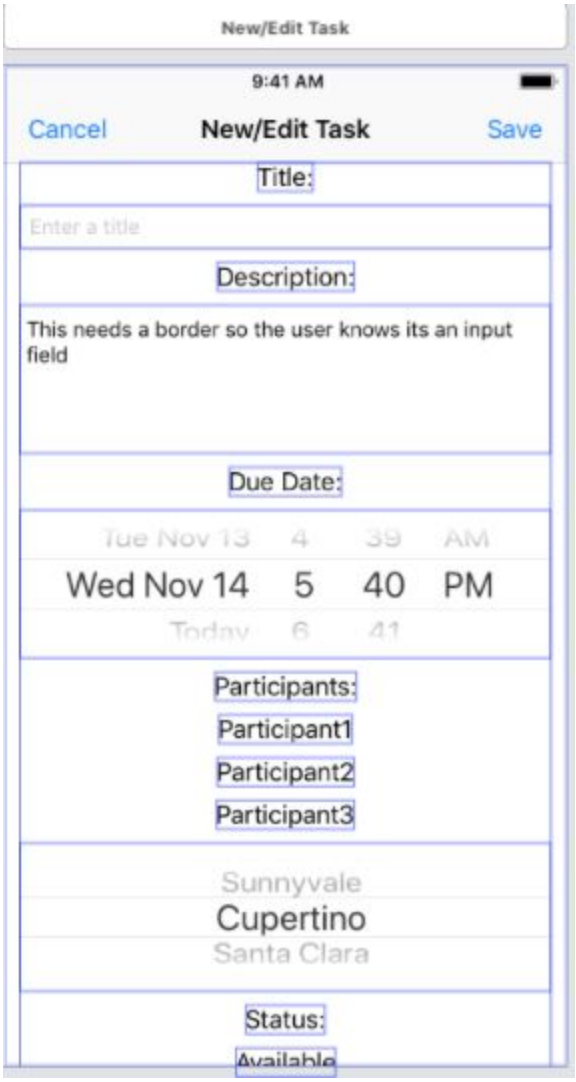



This is a screenshot of the view intended to add or modify a checklist. The view will have two different modes - modify and add mode - such that the same view is reused in either case but in the modify case it is populated with existing values. The main input fields facilitate gathering information from the user to modify the checklist. Certain elements on this page are subject to change, namely the list of participants and the picker at the bottom. The idea is that we are enforcing a limit of three participants per checklist, and these labels may be populated with up to three usernames. The remaining will be hidden. Additionally, the picker will be hidden as well until a "+" button is pressed, which is yet to be added, and the picker will also be hidden when the participant cap is reached. The picker is populated with all users who have signed up for the app. Hitting "Cancel"in the navigation bar navigates back to the view showing the list of checklists a user is involved in without saving any changes to the database, whereas hitting "Save" does the same but will preserve those changes by updating the database. This view has auto-layout constraints that makes the view useable on most devices in portrait orientation, but is not supported for a landscape orientation, and this might be added as a restriction. Currently the view only accepts input and displays the information necessary, but saving or retrieving a checklist from the database has not yet been implemented.

- List Tasks/Delete

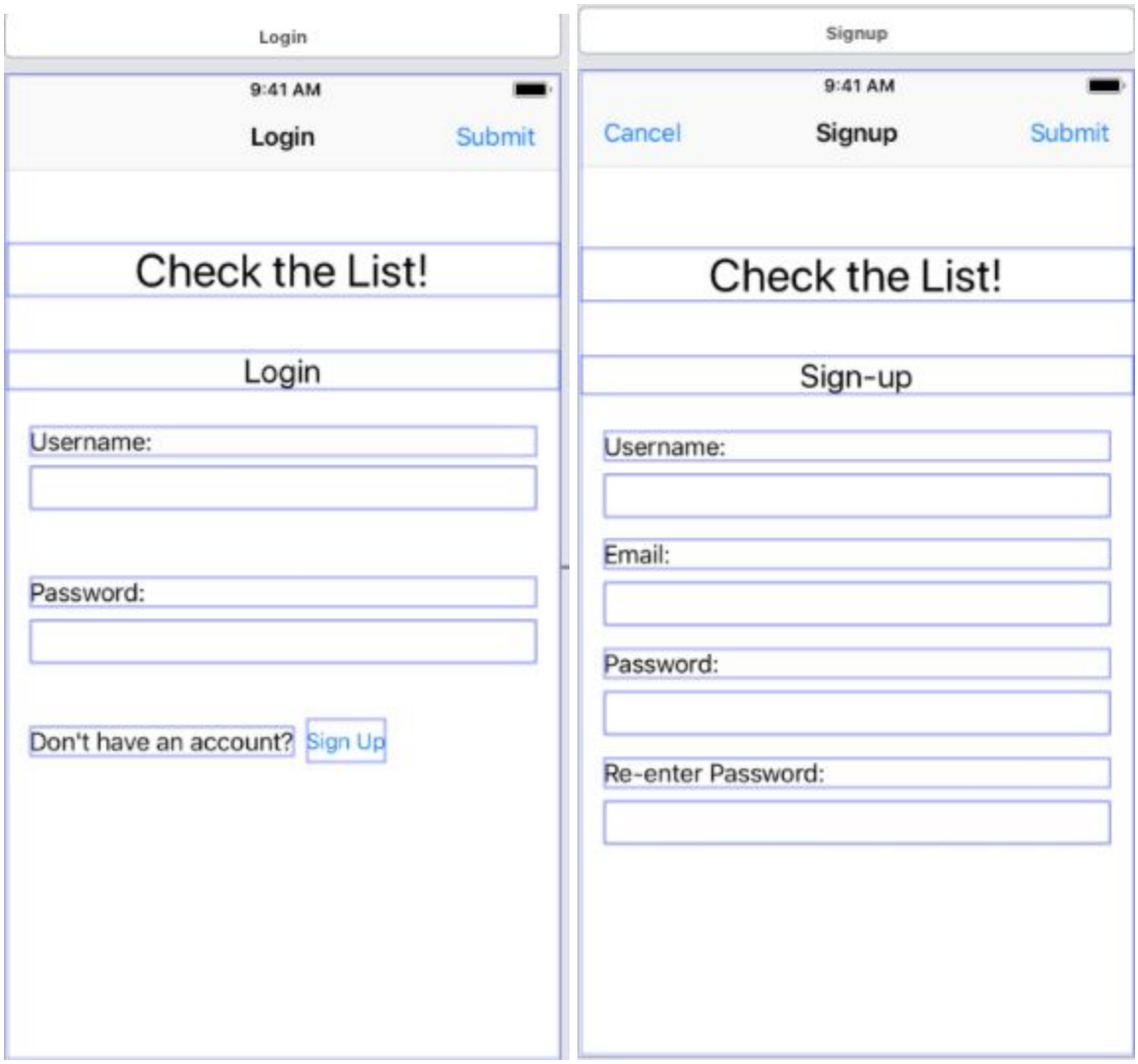

This view functions similarly to the one listing checklists, but its function is to display all the tasks existing for one particularly checklist. At one point we thought we might combine and reuse the view listing the checklists and the view listing the tasks with some sort of condition to determine which information should be shown and hide or show elements accordingly, however I believe that would add unnecessary complication to the code, so I have kept them as separate views. Similar information is displayed, with the behavior for the participants icon, the checkbox image, and the info icon being the same as for the checklist view. The main difference is the presence of a status indicator, which will indicate whether a task is available, in progress, or completed. The navigation works similarly as well, with the "Back" button rewinding to the table view listing all the checklists. The "+" button moves to a view for creating a task item, and tapping on a row outside of an input field will move to a view for editing that specific task item. The "Edit" button functions similarly to the checklist view as well, in using the existing functionality to facilitate the deleting of tasks. The view currently has proper auto-layout constraints to support either portrait or landscape orientation for any device. Currently the view only accepts serves to display what information will be displayed, with some navigation implemented, but the implementation for retrieving the tasks from the database, deleting the tasks, or modifying the cell appearance based on that information has not yet been implemented.

- Create/Modify Task




This view is also quite similar to another view, that being the one for editing or deleting a checklist, but it is intended for use on a task item. Again, it was considered that this view would be reused for checklists and tasks, but the view is already intended to change depending on whether it is a new task or editing an existing one, so making it distinguish between checklist and task as well might be overly complicated for little gain, but that is still something we may look into. The information displayed is similar to the checklist view, except for the status label, which is not an editable field for a new task, as a new task would start with the status “Available”. When this view is accessed in edit mode, the status would become a picker to choose between “Available”, “In progress”, and “Completed”. Additionally, the participants picker will only be populated with the users who have been listed as participants in the parent checklist, rather than all users of the app. The navigation is similar to that for creating/modifying a checklist, with cancel rewinding to the view listing the tasks without making changes, and the save button rewinding while saving any changes to the database. This view has auto-layout constraints that makes the view useable on most devices in portrait orientation, but is not supported for a landscape orientation, and this might be restricted. Currently the view only accepts input and displays the information necessary, but saving or retrieving a task item from the database has not yet been implemented.

- Login or Create User




These views serve to allow logging in and signing up respectively, with input forms requiring basic information needed from the user. The navigation has been implemented between these views, such that clicking “Sign Up” on the Login view moves to the Signup view, and clicking “Cancel” on the Signup view moves back to the Login view. Currently hitting “Submit” on Login does not actually submit any data to the database, but will move directly to the checklist table view without authentication. Hitting “Submit” on the Signup view also does not submit anything to the database yet, but will return to the Login view. Both views have auto-layout constraints that allow them to be viewed on almost any device in portrait orientation, but does not support landscape orientation. The idea is that when implemented, successful login will be required in order to access the rest of the app beyond these two views.

# BackEnd Update

## Firebase and FireStore

This app will use FIrebase and Firestore to save and load data. One of the key functionality of this app is the ability to share the checklists and task among other users and Firestore relatime database allow us to synchronize all the data among all the users in real time without worrying about updating it manually.

Reasons to choose Firebase

- FireStore is a noSql database which allow us to avoid normalization and store data as collections and subcollections which is very optimal for this app.
- FireStore provide .Snapshot() function which allow all the user to get notification right away after change. This saves time for both users and developers as we don't have to worry about regretting it manually.
- Firestore provides .add() and .update() which will allow user to add/edit the same data
- Firebase Auth will allow us to integrate google and other logins easily and Firebase will provide data security.

The app directory have cocoaPods installed which allows it to communicate with google Firebase project. Once the app the deployed, It automatically talks to the database once initialed in app.delegate. A file GoogleService-info.plist allow developers to manage permissions and target from within Xcode.

### Initising the Cloud Firestore

```
import Firebase
FirebaseApp.configure()
let db = Firestore.firestore()
```

### Saving and updating the data.

```
let Checklist: [String: Any] = [
    "ChecklistName": "ChecklistA",
    "Completed": false,
    "totalTask": 3,
```

```
    "userID": “username”,
    "dateCreated": NSDate(),
    "dueDateChecklist": NSDate(),
    "Participants": [“user1”, “user2”, "user3"],
    "priority": “High”,
    "Tasks": [
        "dateCreated": NSDate(),
    "dueDateChecklist": NSDate(),
      "taskName": "taskA",
"Participants": [“user1”, “user2”, "user3"],
"priority": “High”,
"status": “InProgress”,
    ]
]
db.collection("Checklist").document(Checklist.ChecklistName).setData(d
ocData) { err in
    if let err = err {
        print("Error writing document: \(err)")
    } else {
        print("Document successfully written!")
    }
}
```

This will allow us to use setData()  and updateData() to add a new event and edit a previously added. Data can be directly added to the FireStore database from the class within the Xcode project.

## Data Query in realtime

```
db.collection("Checklist").document("checklistOne")
    .addSnapshotListener { documentSnapshot, error in
      guard let document = documentSnapshot else {
        print("Error fetching document: \(error!)")
        return
      }
      guard let data = document.data() else {
        print("Document data was empty.")
        return
```

```
        }
        print("Current data: \(data)")
    }
```

.addSnapshotListener will allow all the user to get the notification as soon as a change is made in the database. This will make our App relatime and update right away.

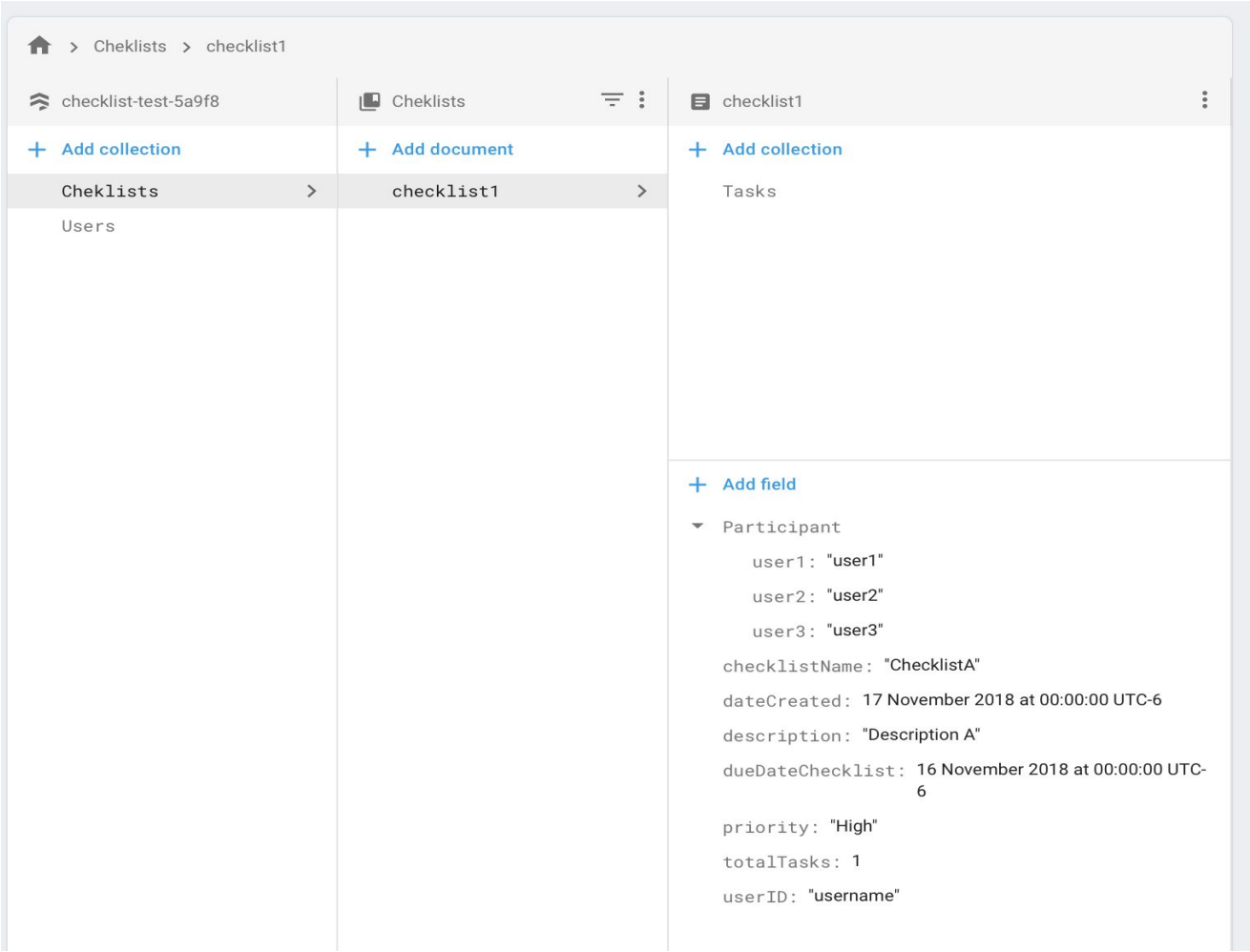
## Offline Data and Data persistence

```
Firestore.firestore().disableNetwork { (error) in
}

Firestore.firestore().enableNetwork { (error) in
}
```

This will allow user to work with app the when offline and push the data automatically when the app goes online.

# Updated Database design



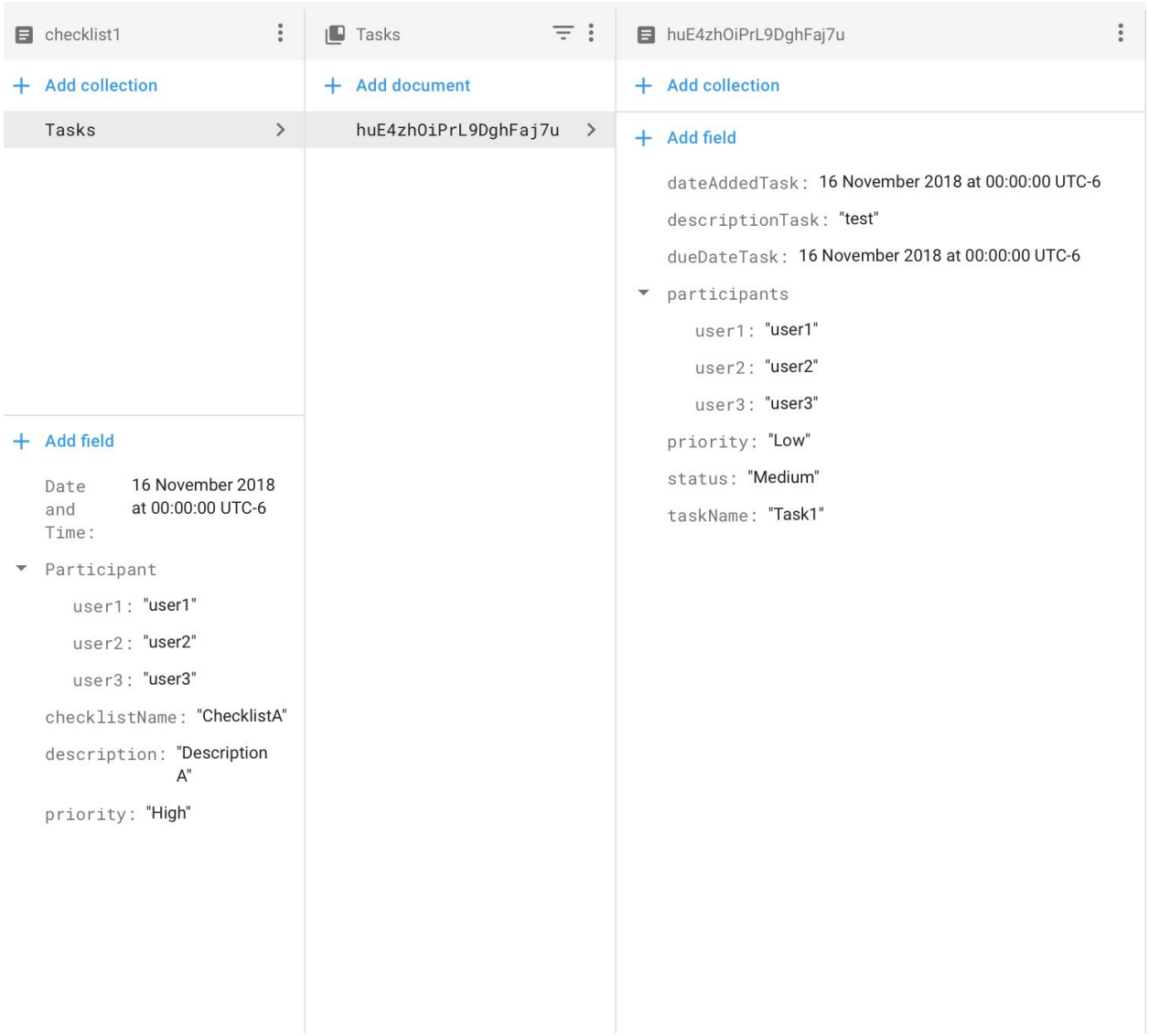

There are two main Collections

- Checklist
- Users

Checklist collection will have the document id as checklist name. This won't allow users to add two checklists with same name and addChecklist and addTask functions will take care of this. Further, each document will have all the information required to add a checklist including userId and total task count.

All the tasks are saved as subcollection in the document of Checklist collection. This might change to Task having their own Collection but tasks are stores as subcollection right now because we never have to query anything in the subcollection "tasks" outside the parent collection and parent document.

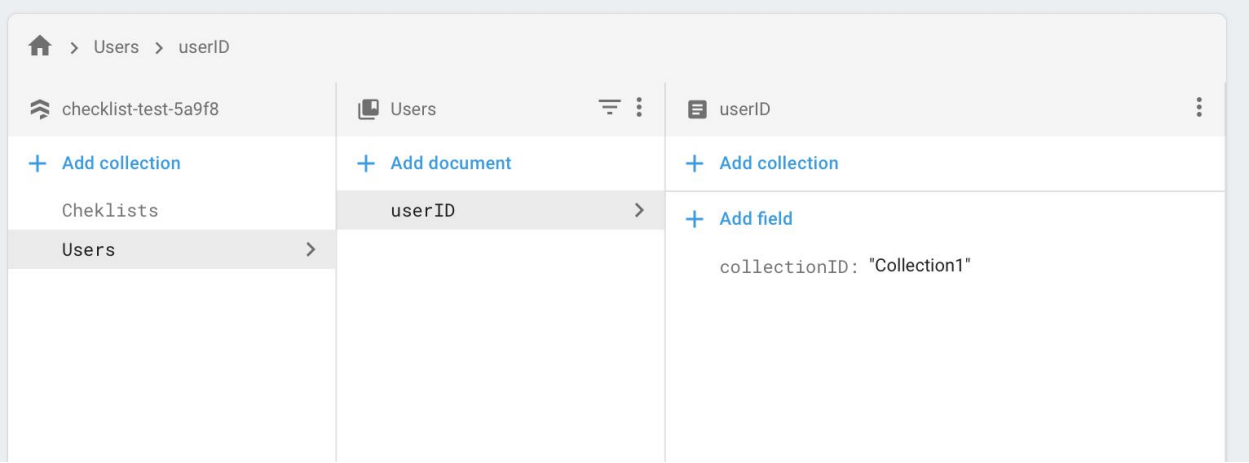
As shown in the picture below, the task will have their autoAssigned id as we never have to query a specific task and only need to load.




The other checklist is the User checklist which shows the link between the Checklist collection and User collection. Firebase allow us to denormalize and data duplicacy is good in nosql database, so saving the shared user in both Checklist Collection and User Collection will help user query faster.
This collections is incomplete and will have future changes such as:

- A new collection to save the User and a collection “Shared” to maintain a many to many relationship between the Checklist and User.

- A boolean entity in both Checklist collection and task sub-collection to keep a track if the task is completed or not.




## Problems, Challenges and Future changes

- The decision to make task a collection or keep it as sub-collection is still pending.
- The relationship between User, Shared and Checklist Collection still needs to be established.
- Pod Firebase/UI will replace the sign in page and users will be able to sign in using google signin/signUp page which will increase security.