Team Name: The Four
Team Members: [Sergio Nogueira Jr. - https://github.com/seno7509] / [Askar Alaskar - https://github.com/moal2839] / [Mohammad Ghiasy - https://github.com/paimang**]**
Project Name: Planneradora
Team Number: 006
Section: 014
GitHub Repository: https://github.com/CSCI-3308-CU-Boulder/3308SP21_section014_6/

Project Milestone 3

- **What project features were completed?**

For our midterm presentation, we were able to create many of the core features that we had originally planned to add to the project. As a group, we were able to finish:

- **User Registration**
  - In-App Registration
    - Created a class for users, grabbed fields from the textboxes and implemented them into a new instance of the user class (email & password)
  - In-Database insertion
    - Inserted the new instance of the user class model into FireBase
- **User Sign-in**
  - In-App Sign-In
    - Create new model of user class, sent it to firebase
  - In-Database SQL Query
    - Use SQL (SQLite) query in firebase to find if the user exists.
- **Add Task**
  - Functionality to add a new task to a listview that is displayed within the application.
  - Created a task class/model to store strings for the specifics of the task
    - Difficulty
    - Name of Task
    - Due Date
    - Description
- **Delete Task**
  - Mapped deletion function to bind to holdClick() event listener on the listview. Meaning that when someone in the app clicks and holds a task, the task will be deleted.

- **Update List**
    - Created a button and the functionality therein to refresh the list of tasks by pressing on an XML button that activates a function in the onClick() java mapping.
- **DateTime Display**
    - Created a variation of textviews with different formats in XML that updates the current date and time and displays them in the main_activity.xml display.

- **What worked during the Demo?**

All of the aforementioned tasks above worked during the demo. We had a bit of difficulty with new task additions interfering with previous task additions (new tasks would be written on top of old tasks *visually* despite still being stored in the back-end code of the program). Proper event handlers for creation of accounts that already exist in the database were added, as well as credential checks that needed to be fulfilled before the SQL insertion/select queries were ever called. The deletion of tasks by holding was fully functional, as well as the updating function, registration, datetime display, and sign-in.

- **What issues were faced during the development or during the Demo?**

During the demo, we had an issue with the addTask() function interfering with XML display in properly showing previously-added tasks. In development, we had a whole array of difficulties: The use of SQLite within android studio created a whole slew of problems where the queryString that was used in the DatabaseHelper would not be interpreted correctly by SQLite, creating an incredible setback in the development of the application. The switch to Firebase expedited this process immensely as the Firebase mAuth API had built-in Java functions that were exponentially easier to assign queries to. They even had a built-in query function to search for a simple java email/password whereby it translated it to a SQL script. We also had a whole array of problems when switching views to and from the addTask() function that would call the XML view to inherit the display hierarchy, meaning that we had to use the built-in emulator's 'undo' arrow to navigate out of the addTask modal.

- **What were the biggest suggestions offered by the TA?**

Shruthi suggested that when we are ready to transition the tasks in the application to be stored in the database, that we should use the custom ID for each user and tie it to another value that would be an array of Tasks. Each user would have an array attached to their ID to prevent back-end access to user credentials while still knowing which user the tasks were for. From there, we would initialize the taskList on account creation, and properly append/find/delete tasks

within the user's data. Shruthi suggested that we also streamline a lot of the other front-end displays of our application (stretch images to fit screen better, better fonts, better modal functionality and transitions, longer confirmation/error messages so that the user has time to read them).

- **Individual Contributions by each team member**

Sergio Nogueira: I created our FireBase database as well as optimized it to function with our java code. The database accepts an email and password and uses the FireBase google API to verify that the email is a valid email. I added back-end code that prevents the user from entering empty passwords/usernames and passwords that are under 6 characters. This prevented the application from crashing if the database helper functions sent empty strings to FireBase. I added a few different Toast messages to pop up depending on the user's mistake, in order to best articulate the solution to the user. I created the Java code for the registration/sign-in functions that accept textbox text contents, convert them to string text, and trim whitespace, then send those string values to the UserModel class, and call the SQL helper functions that send the UserModel to Firebase. I created the front, back-end, and database connection to registration & sign-in.

JIRA: https://csci-3308-spring21-6.atlassian.net/jira/people/5f5b8844e0d3060076653740

Askar Alaskar: I implemented the add task button and the functionality of updating and removing tasks. The add task button on the main page of the application once clicked on lands you on a different xml page that prompts the user to enter information about the new task that they are going to add. Once done entering all the information about the task there is a plus sign button at the bottom of the page that once clicked adds this information to some global variables for now that will update the task list for the homepage for now the plan is to integrate the task class into the firebase database so that clicking the button updates it in the database and not to some global variables. The user then can hit the back button on the top left of the page and go back to the main page where they can press the update button that will refresh the task list and make them visible to the screen showing what tasks that the user has. The current problem with this as of the moment is that because global variables are being used to store information about the tasks they get overwritten every time a new task is added that will be fixed once tasks are integrated to the database. Once a task is visible on the main page the user on hold on that task or long press it and it will remove it. We are thinking of giving a user in the future some kind of message when the long press a task to ask whether they want to delete it or modify it and then confirm their decision. One further idea is also to implement just a simple click without holding

and that will pull up a view of the task where the user can view all the details about the task and also click a check box button to check the task as being done.

Payman Ghiasy: I have implemented our U-I by lot but have not pushed it into github because I wanna make sure that my team agrees with what I did, I created another xml also to has design of how the task will appear in the list and also a page where we can show all the tasks that we will be getting from data (we are working on it right now), I have also created another page for calendar when somebody creating a task and click on date , xml will pop up that will allow user to choose a date throw that and once it's done it will automatically appear in the box. But none of these implementations are pushed yet because we are still very new to android studio and don't wanna cost any conflict that will miss up somebody elses work.