ShareList

Hannan Khan
Yanzhi Wang
Anum Farrukh Khan
Mohammed Furkhan Shaikh
Yu Zhao

Group 6 - Coffee_Code
Iteration 2
04-07-2021
CSE 5324
Software Engineering 1
Analysis, Design, Testing

Agenda

1. Project Description
2. Overall Design Approach
3. Requirements
4. Use Cases
5. High Level Use Cases
6. Use Case Diagram
7. RUCTM
8. Increment Matrix
9. Expanded Use Cases
10. Updated Domain Diagram
11. Sequence Diagrams
12. Updated Task List
13. Design Class Diagram
14. Code Snapshots
15. Summary

Project Description

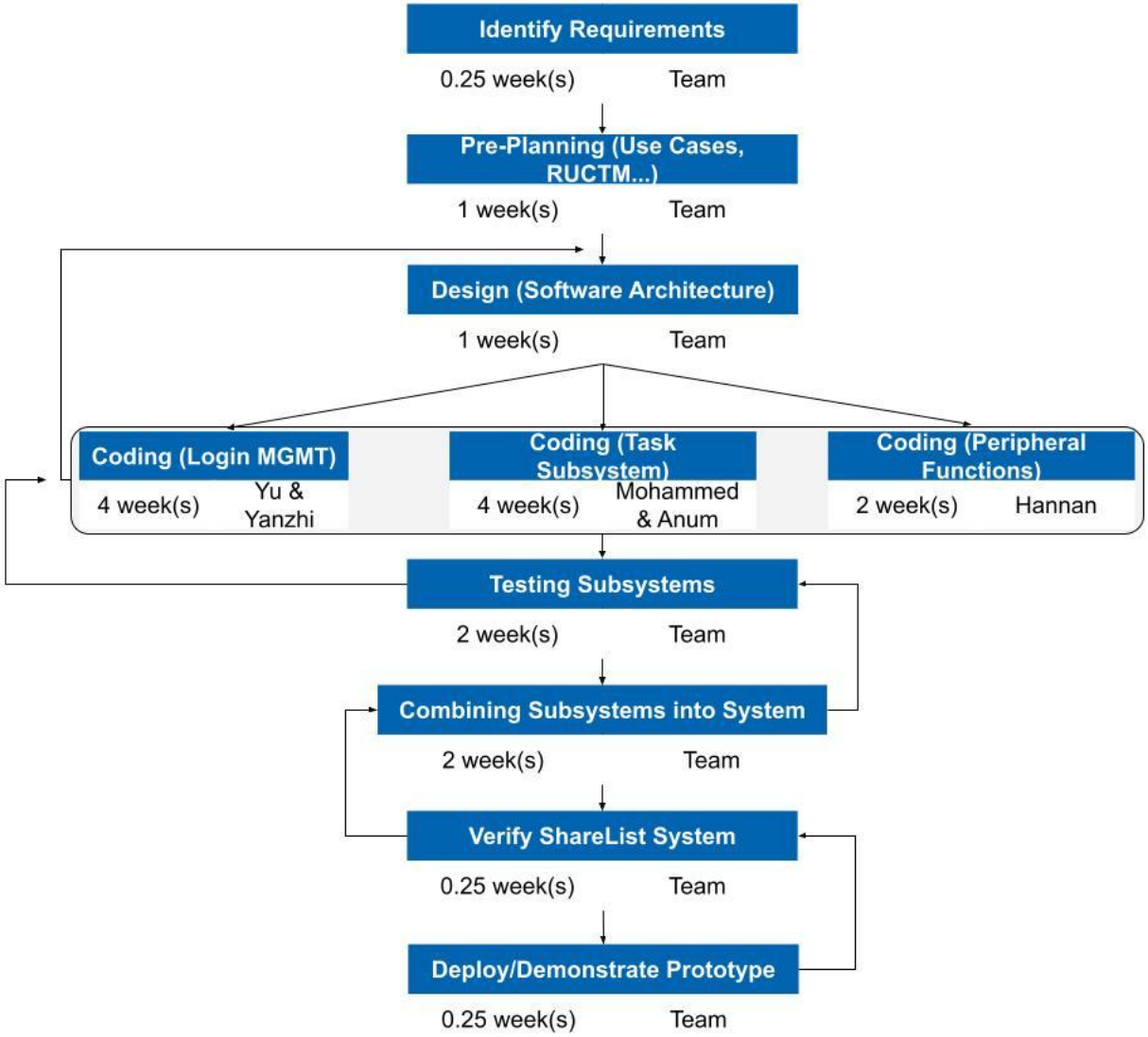
ShareList is designed to allow users the ability to create, delegate, and track tasks that are shared among users in a group.

Our goal is to allow teams in virtual workspaces the ability to have better collaboration via shared/delegated tasks.

Think of each group as a 'project' in which members contribute towards the tasks they are assigned. It is a similar concept to a Whatsapp group. The messaging is replaced with tasks that can have users assigned to them.

Features include: group admins, group summary reports, database storage, and wireless synchronization of database.

Overall Design Approach



Requirements

Req ID	Req Statement	Line #
R1	The system shall provide for authenticated login to the ShareList app.	
R1.1	Register.	40
R1.2	Login.	
R1.3	Log Off.	
R1.4	Reset password.	
R2	The system shall provide admins the ability to:	
R2.1	Add tasks.	41,42
R2.2	Remove tasks.	
R2.3	Edit tasks.	
R2.4	Assign tasks.	
R2.5	Add users to group.	46,47
R2.6	Remove users from the group.	34,35
R2.7	Add new Admins to group.	18,47
R2.8	Delete group	48,49
R2.9	Edit group.	46,47
R3	The system shall provide users/admins the ability to:	
R3.1	Mark task as completed.	44,45
R3.2	Create groups.	46,47
R4	The system shall provide the ability to search by keywords.	43
R5	The system shall provide tasks the ability to:	
R5.1	Be visible to all members in that group.	22,23
R5.2	Have users assigned to it.	41,42
R5.3	Have assigned users mark it as completed.	44,45
R6	The system shall provide tasks the following properties: start time, completion time, due date, time spent, title, description, users assigned	50
R7	The system shall provide users the ability to create groups, and have admin status when they do.	46,47
R8	The system shall provide a database used for storing information related to tasks, groups, users.	59
R9	The system shall provide wireless resources used for syncing between each user's device and the database.	60
R10	The system shall provide a group summary function, which will accomplish:	50
R10.1	Calculating and reporting the tasks completed.	52
R10.2	Calculating and reporting the in-progress tasks.	53
R10.3	Calculating and reporting the tasks not completed by due date.	54

Use Cases

Use Case #	Use Case Name
UC1	Login
UC1.1	Register.
UC1.2	Login.
UC1.3	Log Off.
UC1.4	Reset password.
UC2	Manage task
UC2.1	Add tasks.
UC2.2	Remove tasks.
UC2.3	Edit tasks.
UC2.4	Assign tasks.
UC3	Manage group
UC3.1	Add users.
UC3.2	Remove users.
UC3.3	Add admins.
UC3.4	Delete groups
UC3.5	Edit groups.
UC4	Mark task as completed
UC5	Create groups.
UC6	Search by keywords.
UC7	Generate summary

High Level Use Cases

UC 1.1: Register
- TUCBW the user entering required details and clicking the register button.
- TUCEW the user being registered successfully.

UC 1.2: Login
- TUCBW the user enters the username and password.
- TUCEW the user successfully logs in and can see the home page.

UC 1.3: Log Off
- TUCBW the user clicks on the logout button.
- TUCEW the user is successfully logged out and can see the login page.

UC 1.4: Reset Password
- TUCBW the user clicks on the reset password option.
- TUCEW the password of the user is changed.

UC 2.1: Add Tasks
- TUCBW the admin clicking on the “Add Tasks” option.
- TUCEW the admin adding the task successfully and can see the added tasks.

UC 2.2: Remove Tasks
- TUCBW the admin clicking the “Remove Task” option.
- TUCEW the admin removing the task successfully.

UC 2.3: Edit Tasks
- TUCBW the admin clicking on the “Edit Task” option and making the necessary changes.
- TUCEW the admin being able to see the changes made to the respective tasks.

UC 2.4: Assign Tasks
- TUCBW the Admin assigning the task to particular users.
- TUCEW the users get assigned new tasks.

UC 3.1: Add users
- TUCBW the admin clicking on the add member option.
- TUCEW the user being added to the selected group.

UC 3.2: Remove users
- TUCBW the admin selecting a user to be removed from the group.
- TUCEW the user is no longer a member of the group.

UC 3.3: Add admins
- TUCBW the admin selecting a member to be an admin.
- TUCEW the selected user being the admin for the group.

UC 3.4: Delete Group
- TUCBW the admin selecting the delete group option.
- TUCEW the group, related tasks and users, being removed from the records.

UC 3.5: Edit Group
- TUCBW the admin selecting the edit group option.
- TUCEW the group information is updated by the admin.

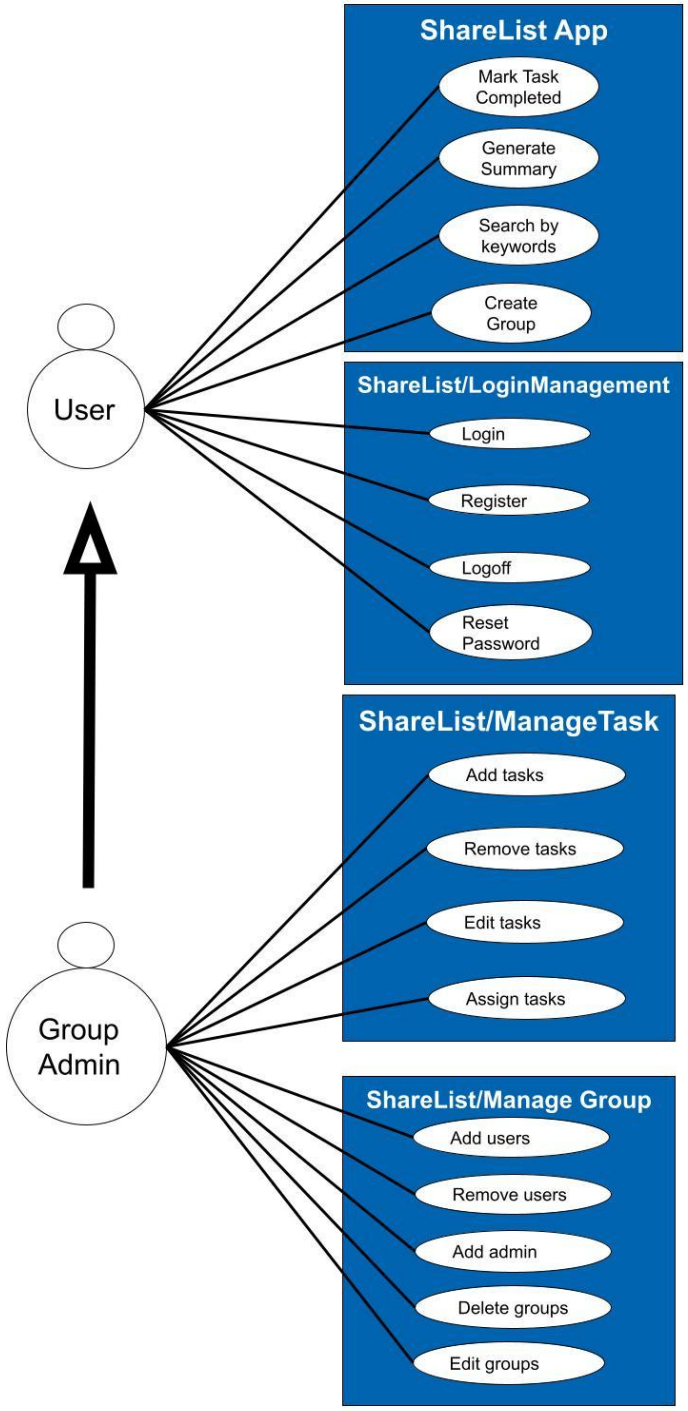
UC 4: Mark task as completed
- TUCBW the users selecting their assigned tasks to be marked complete.
- TUCEW the tasks information being updated by the users.

UC 5: Create Group
- TUCBW the user selecting the create group option.
- TUCEW new group being created with the user as the admin of the group.

UC 6: Search by keywords
- TUCBW the user clicking on the search option.
- TUCEW the search results being returned to the user.

UC 7: Generate summary
- TUCBW the user selecting the generate summary option.
- TUCEW the summary of tasks and reports being displayed to the user.

Use Case Diagram



RUCTM

Req ID	Priority Weight	UC 1	UC 1.1	UC 1.2	UC 1.3	UC 1.4	UC 2	UC 2.1	UC 2.2	UC 2.3	UC 2.4	UC 3	UC 3.1	UC 3.2	UC 3.3	UC 3.4	UC 3.5	UC 4	UC 5	UC 6	UC 7
R1	3	X																			
R1.1			X																		
R1.2				X																	
R1.3					X																
R1.4						X															
R2	1						X														
R2.1								X													
R2.2									X												
R2.3										X											
R2.4											X										
R2.5													X								
R2.6														X							
R2.7															X						
R2.8																X					
R2.9																	X				
R3																					
R3.1																		X			
R3.2																			X		
R4	6																			X	

RUCTM

Req ID	Priority Weight	UC 1	UC 1.1	UC 1.2	UC 1.3	UC 1.4	UC 2	UC 2.1	UC 2.2	UC 2.3	UC 2.4	UC 3	UC 3.1	UC 3.2	UC 3.3	UC 3.4	UC 3.5	UC 4	UC 5	UC 6	UC 7
R5	1						X														
R5.1								X													
R5.2									X												
R5.3										X											
R6																					
R7	1											X									
R8	2																				
R9	4																				
R10	5																				X
R10.1																					X
R10.2																					X
R10.3																					X
Score		3	3	3	3	3	2	2	2	2	1	1	1	1	1	1	1	1	1	7	24
Legend	1 is highest priority.	Work 1 first.																			

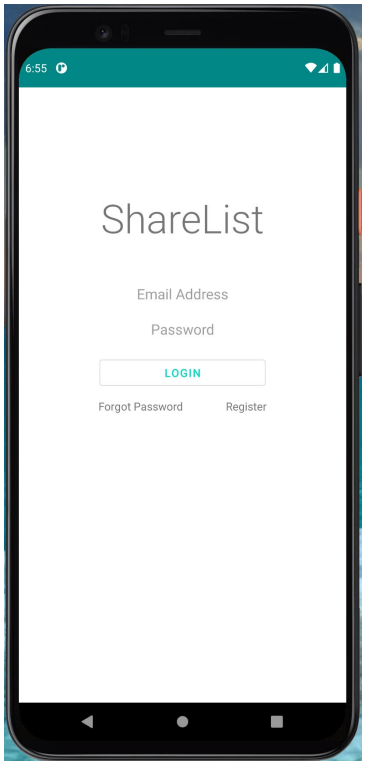
Increment Matrix

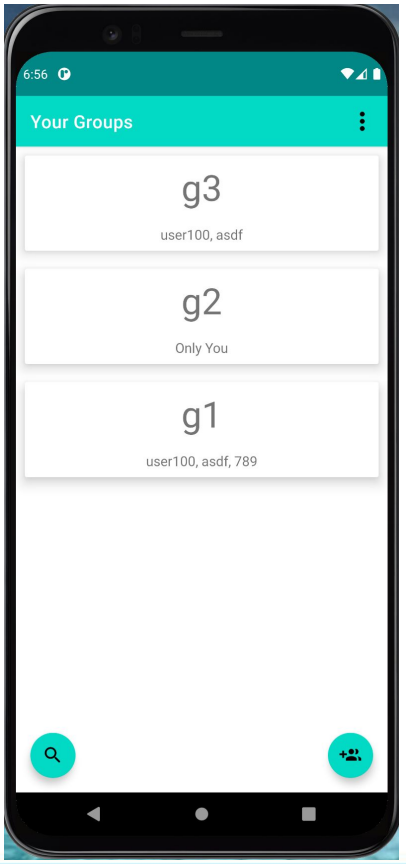
Use Case #	Priority	Effort (Person-weeks)	Depends on	Iteration 1 (03/12)	Iteration 2 (04/09)	Iteration 3 (05/03)
UC1						
UC1.1	3	6	None		6	
UC1.2		6			6	
UC1.3		6			6	
UC1.4		6				6
UC2						
UC2.1	2	2	None	2		
UC2.2		2				2
UC2.3		2			2	
UC2.4		3			3	
UC3						
UC3.1	1	4	UC 1		4	
UC3.2		4			4	
UC3.3		4				4
UC3.4		4				4
UC3.5		4				4
UC4		3	UC 2		3	
UC5		4	UC 3		4	
UC6	5	2	UC 2		2	
UC7	4	1				2
Total Effort		63		2	40	22
1 PW = 5 hrs				10	200	110

Expanded Use Case - Login

EUC1.2: Login

Precondition: *This use case assumes that the user is seeing the Login page and is a registered user.*	
Actor: User	System: ShareList
1. TUCBW the user enters the details and clicks on the Login button. 3) TUCEW the user successfully logs in.	0. System displays the Login page. *2. System checks the credentials entered by user: a) If the credentials are correct, the users 'Your Groups' page will be displayed. b) If incorrect, the system will display an “Incorrect username/password” message.
Postcondition: *The user will be able to access the features of the app.*	

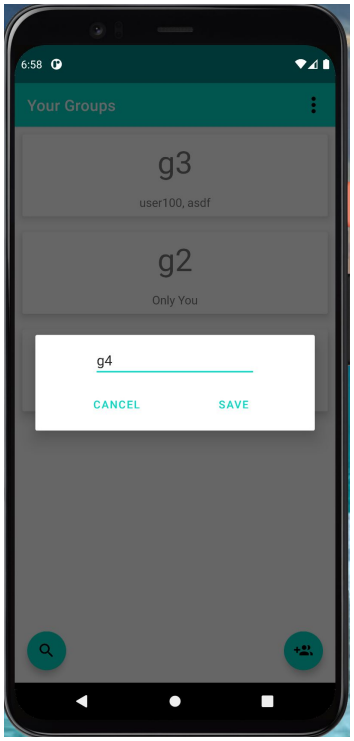


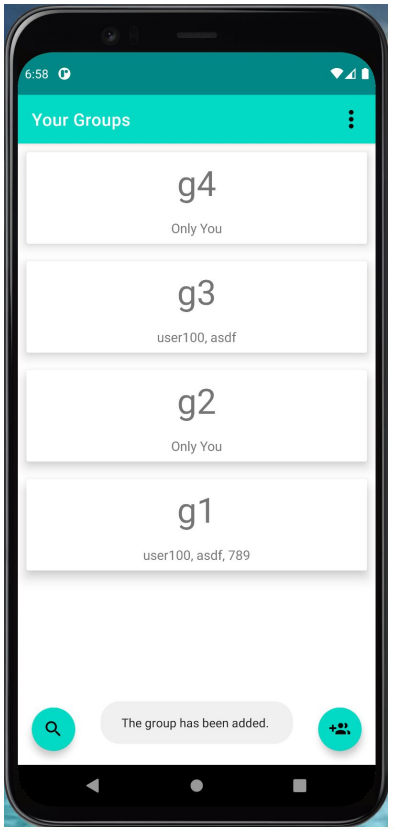


Expanded Use Case - Create Group

EUC5: Create Group

Precondition: *This use case requires that the user is logged in to the ShareList App, and seeing the 'Create Group' activity.*	
Actor: User	System: ShareList
1. TUCBW user clicking on the Create Group button. 3. The user fills in the text field and clicks the save button. 5. TUCEW the user sees a confirmation “The group has been created”.	0. System displays the Create Group activity. 2. System displays text field to enter group name. *4. System checks for errors and processes the save request.
Postcondition: *The group is created, stored in the database, and available to be acted upon immediately.*	

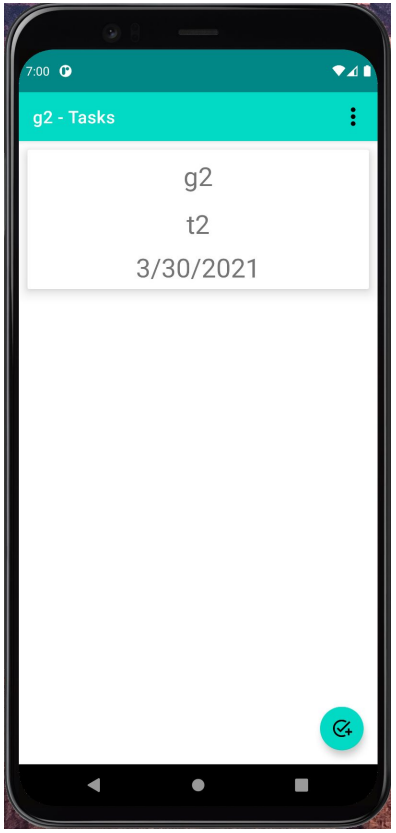


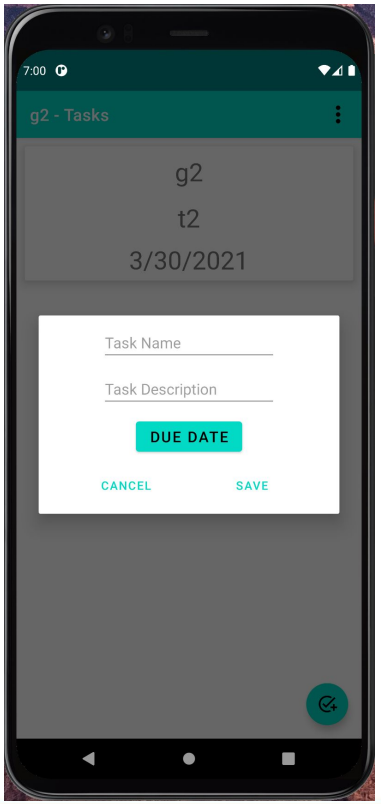


Expanded Use Case - Add Task

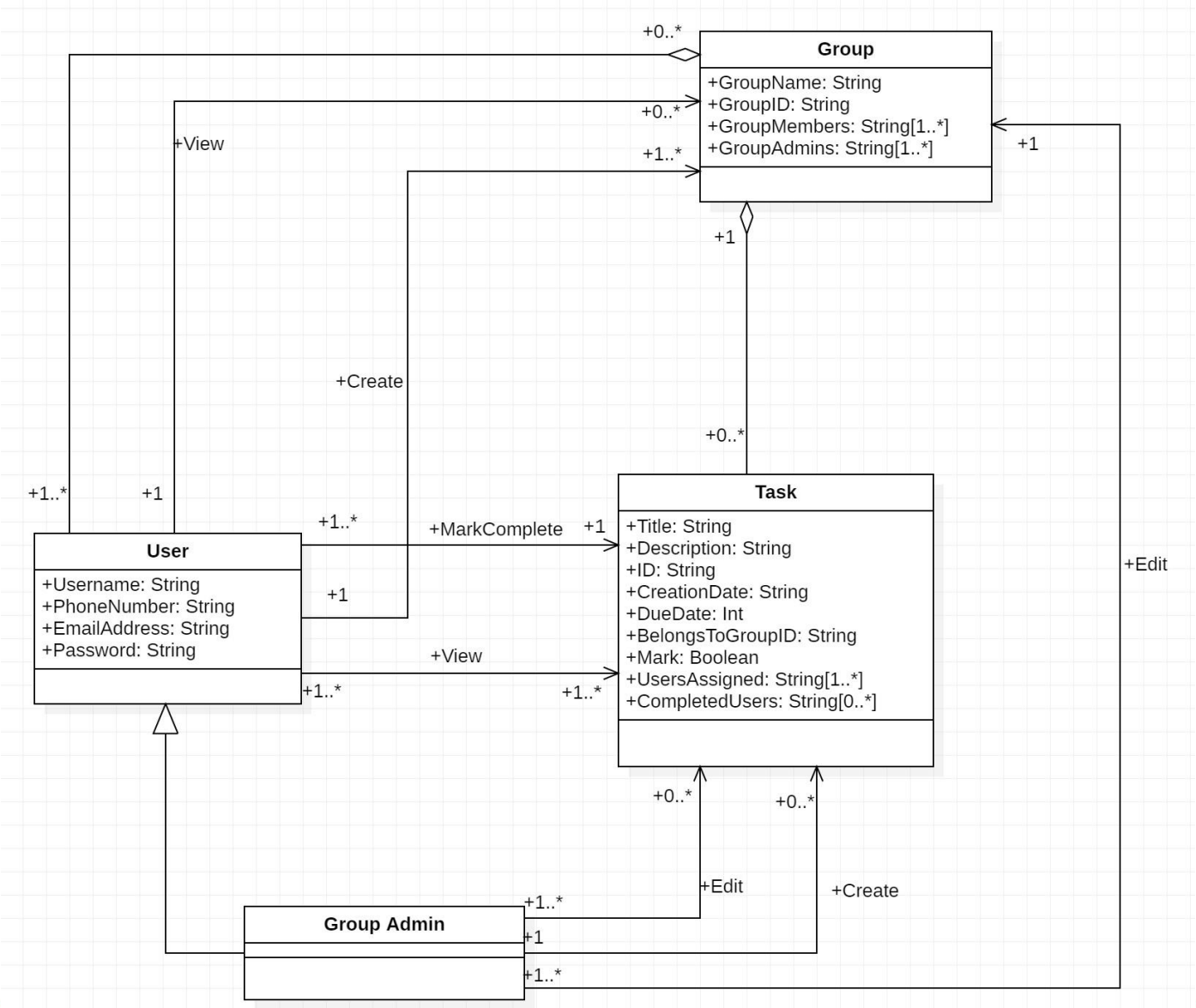
EUC2.1: Add Tasks

Precondition: *This use case requires that the user is logged in and is an admin of the group.*	
Actor: Group Admin	System: ShareList
1. TUCBW group admin clicking on the Add Task button. 3. The group admin enters the information and clicks on the Save button. 5. TUCEW the group admin sees a confirmation “The task has been added” on the app	0. System displays the Add task button. 2. System prompts the group admin to enter the Task name, Task Description and Due date. *4. System checks for errors, and processes the save request.
Postcondition: *The task is created successfully and stored in the database.*	

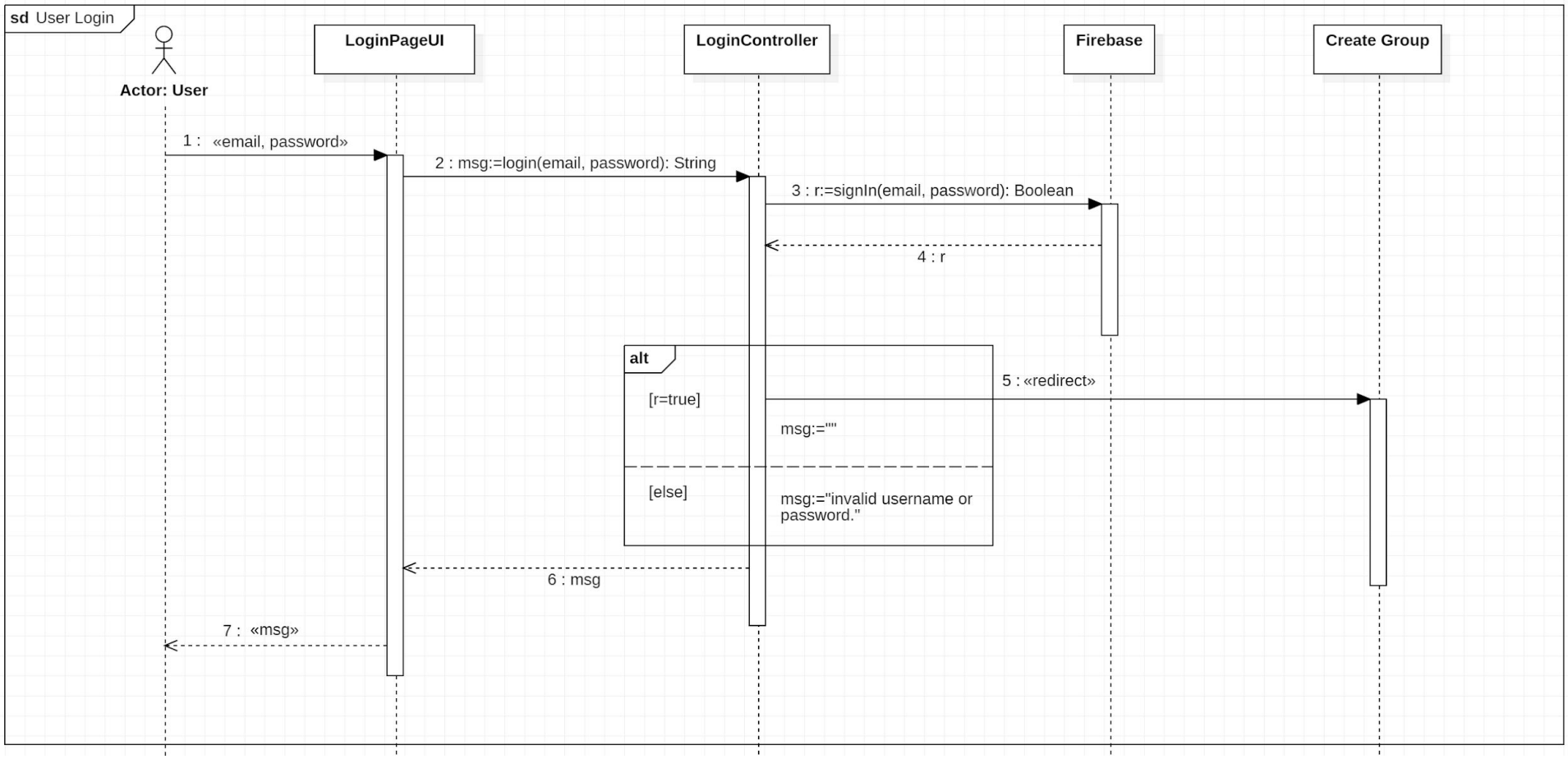




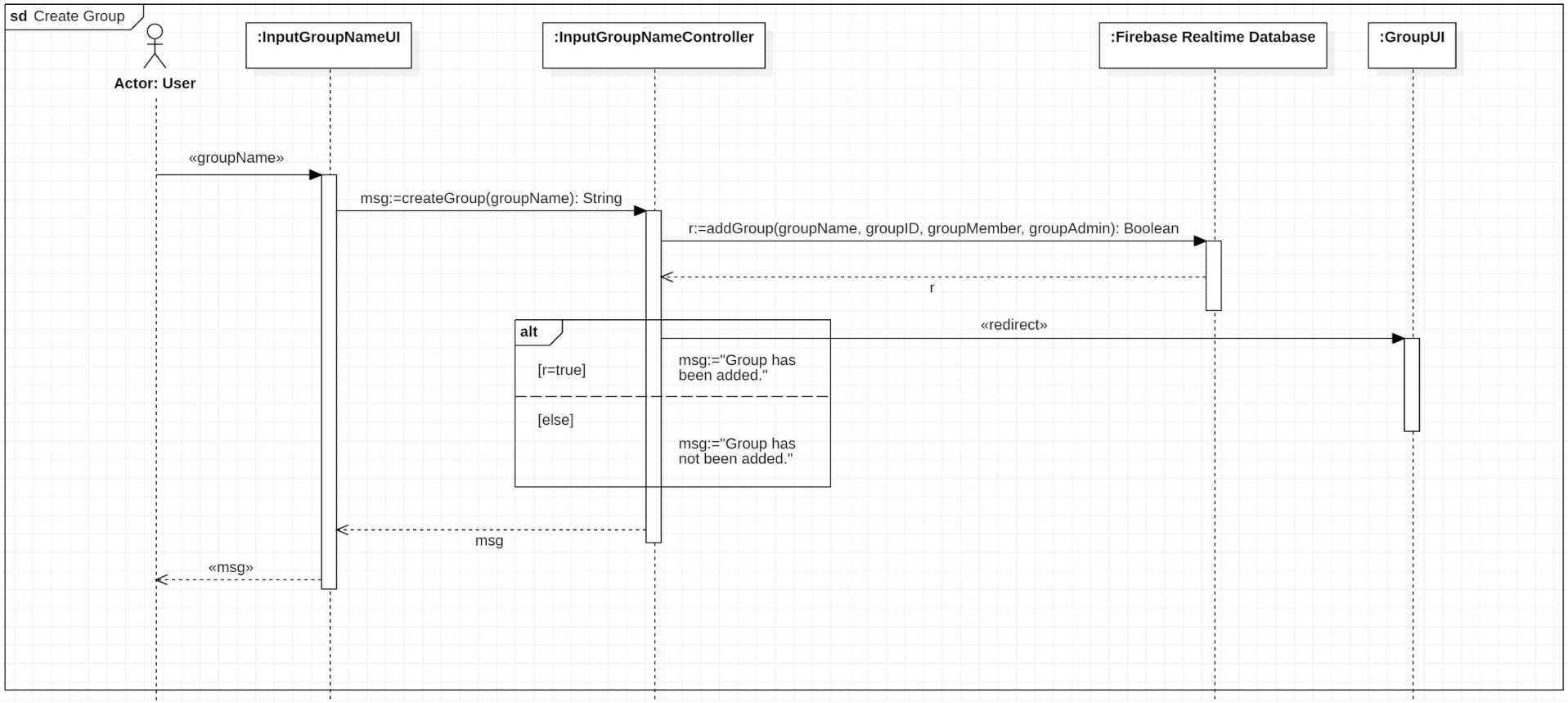
Updated Domain Diagram



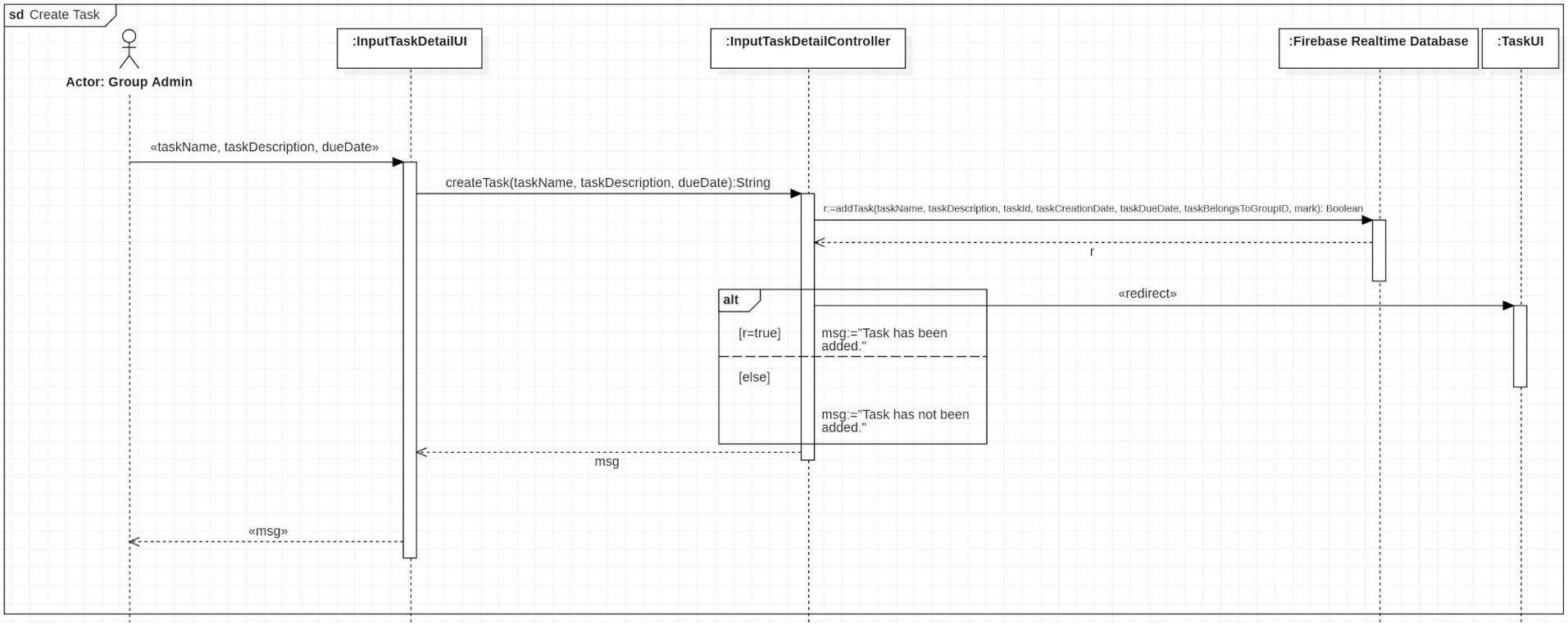
Sequence Diagram - Login



Sequence Diagram - Create Group



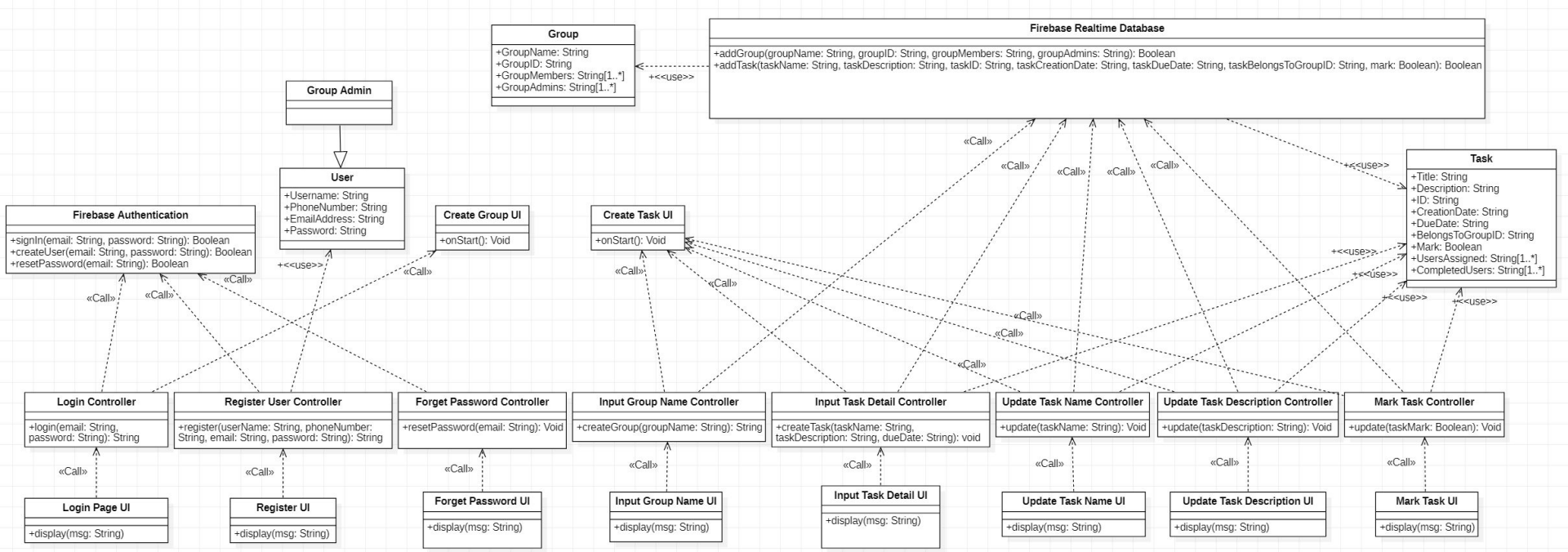
Sequence Diagram - Add Task



Updated Task List

Assigned to...	Iteration 2	Iteration 3	Use Case
Mohammed & Anum	Add tasks.		2
		Remove tasks.	2
Yanzhi & Yu	Register.		1
	Login.		1
Mohammed & Anum	Manage task.		2
	Edit tasks.		2
Hannan	Mark task as completed.		4
	Create groups.		5
	Search by keywords.		6
		Generate summary.	7
Yanzhi & Yu	Log Off.		1
		Reset password.	1
Mohammed & Anum	Assign tasks.		3
	Manage group.		3
	Add users.		3
	Remove users.		3
		Add admins.	3
		Delete groups.	3
		Edit groups.	3

Design Class Diagram



Code Snapshot - Login

```
/**
 * This method gets the email and password and uses Firebase to sign-in the user.
 * Displays a toast if it fails.
 */
private void loginActivity() {
    // everything will be converted to string
    String email = loginEmail.getText().toString().trim();
    String password = loginPassword.getText().toString().trim();

    // validate that everything is not empty
    if (email.isEmpty()) {
        loginEmail.setError("It should not be empty. ");
        loginEmail.requestFocus();
        return;
    }
    if (password.isEmpty()) {
        loginPassword.setError("It should not be empty. ");
        loginPassword.requestFocus();
        return;
    }

    // connect to the firebase
    // The user will use their email and password to login
    auth.signInWithEmailAndPassword(email, password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            // If the user login, the user will go to the create group activity
            if (task.isSuccessful()) {
                // Clears the password for if you logout.
                loginPassword.setText("");
                startActivity(new Intent( packageContext: MainActivity.this, CreateGroup.class));
            } else {
                Toast.makeText( context: MainActivity.this,  text: "Incorrect Email or Password!", Toast.LENGTH_LONG).show();
            }
        }
    });
}
```

Code Snapshot - Create Group

```java
/**
 * Creates an alert dialog to get information about what group to create. This method will then
 * create the group and set the group admin as the current user. It will also add the current
 * user to the group members hash map.
 * All of this info gets updated in the database, with a toast message signaling success.
 */
private void addGroupActivity() {
    AlertDialog.Builder alertDialog = new AlertDialog.Builder( context: this);
    LayoutInflater layoutInflater = LayoutInflater.from(this);

    View view = layoutInflater.inflate(R.layout.activity_input_group_detail, root: null);
    alertDialog.setView(view);

    AlertDialog dialog = alertDialog.create();
    dialog.setCancelable(false);

    EditText groupName = view.findViewById(R.id.addGroupName);
    Button groupSaveButton = view.findViewById(R.id.groupSaveButton);
    groupSaveButton.setOnClickListener((v) -> {
        // Everything is converted to string
        String groupNameStr = groupName.getText().toString().trim();
        // The id generated here is an ID for the group we will create.
        String id = databaseReferenceGroup.push().getKey();
        Map<String, String> groupMembers = new HashMap<>();
        Map<String, String> groupAdmins = new HashMap<>();

        // Validate everything is not empty
        if (groupNameStr.isEmpty()) {
            groupName.setError("Group name cannot be empty. ");
            return;
        } else {
            Group group = new Group(groupNameStr, id, groupMembers, groupAdmins);
            group.addGroupMember(currUserID);
            group.addGroupAdmin(currUserID);
              src: https://stackoverflow.com/questions/39815117/add-an-item-to-a-list-in-firebase-database
              I have started re-iterating to myself: whenever you find yourself doing
              array.contains("xyz"), you should probably be using a set instead of an array.
              The above mapping with "key": true is an implementation of a set on Firebase.

            databaseReferenceGroup.child(id).setValue(group).addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
                    Toast.makeText( context: CreateGroup.this, text: "The group has been added. ", Toast.LENGTH_LONG).show();
                } else {
                    Toast.makeText( context: CreateGroup.this, text: "The group has not been added. ", Toast.LENGTH_LONG).show();
                }
                dialog.dismiss();
            });
        }
    });

    Button cancelGroupButton = view.findViewById(R.id.groupCancelButton);
    cancelGroupButton.setOnClickListener((v -> dialog.dismiss()));

    dialog.show();
```

Code Snapshot - Add Task

```java
/**
 * Deals with the Task Button. Creates an alert dialog to show the user fields to input the task
 * details.
 * When `save` is pressed, this method will create a task with the contents of the alert dialog
 * and will update the database to match. A toast message is displayed on success.
 */
private void addTaskActivity() {
    AlertDialog.Builder alertDialog = new AlertDialog.Builder( context: this);
    LayoutInflater layoutInflater = LayoutInflater.from(this);

    // Inflate the input task activity
    View view = layoutInflater.inflate(R.layout.activity_input_task_detai
    alertDialog.setView(view);

    AlertDialog dialog = alertDialog.create();
    dialog.setCancelable(false);

    EditText taskName = view.findViewById(R.id.addTaskName);
    EditText taskDescription = view.findViewById(R.id.addTaskDescription)
    Button taskDueDate = view.findViewById(R.id.addTaskDueDate);

    taskDueDate.setOnClickListener((v) -> ShowDatePickerDialog(taskDueDat

    // Task save button
    Button taskSaveButton = view.findViewById(R.id.taskSaveButton);
```

```java
// We can use the statement lambda to make the code easier to understand
taskSaveButton.setOnClickListener((v) -> {
    // Everything is converted to string
    String taskNameStr = taskName.getText().toString().trim();
    String taskDescriptionStr = taskDescription.getText().toString().trim();
    // This ID is a new ID created for the task we are about to store in the database.
    String id = databaseReference.push().getKey();
    String creationDate = DateFormat.getDateInstance().format(new Date());
    String dueDateStr = taskDueDate.getText().toString().trim();

    // Validate that everything is not empty
    if (taskNameStr.isEmpty()) {
        taskName.setError("It should not be empty. ");
        return;
    } else if (taskDescriptionStr.isEmpty()) {
        taskDescription.setError("It should not be empty. ");
        return;
    } else if (dueDateStr.isEmpty()) {
        taskDueDate.setError("It should not be empty. ");
        return;
    }

    Task task = new Task(taskNameStr, taskDescriptionStr, id, creationDate, dueDateStr,  mark: false, groupIDStr);
    Log.d(TAG,  msg: "groupIDStr is " + groupIDStr);
    databaseReference.child(id).setValue(task).addOnCompleteListener(task1 -> {
        if (task1.isSuccessful()) {
            Toast.makeText( context: CreateTaskAdmin.this,  text: "The task has been added. ", Toast.LENGTH_LONG).show();
            dialog.dismiss();
        } else {
            Toast.makeText( context: CreateTaskAdmin.this,  text: "The task has not been added. ", Toast.LENGTH_LONG).show();
            dialog.dismiss();
        }
    });
});

Button cancelTaskButton = view.findViewById(R.id.taskCancelButton);
cancelTaskButton.setOnClickListener((v -> dialog.dismiss()));

dialog.show();
```

Summary

Today we discussed the ShareList App

- How it enables better collaboration between users.
- The requirements and implementation approach.
- Any updates to our app development.
- Expanded UCs/User Interface Prototypes
- Updated Domain Diagram/Sequence Diagrams
- Updated Task List
- Design Class Diagram
- Brief Code Analysis

UNIVERSITY OF TEXAS ARLINGTON

Thank You