Capstone Project: Task Scheduling Application

Michael Lawson

Colorado State University, Global

CSC480: Computer Science Capstone

Prof. Chintan Thakkar

March 10, 2024

Capstone Project: Task Scheduling Application

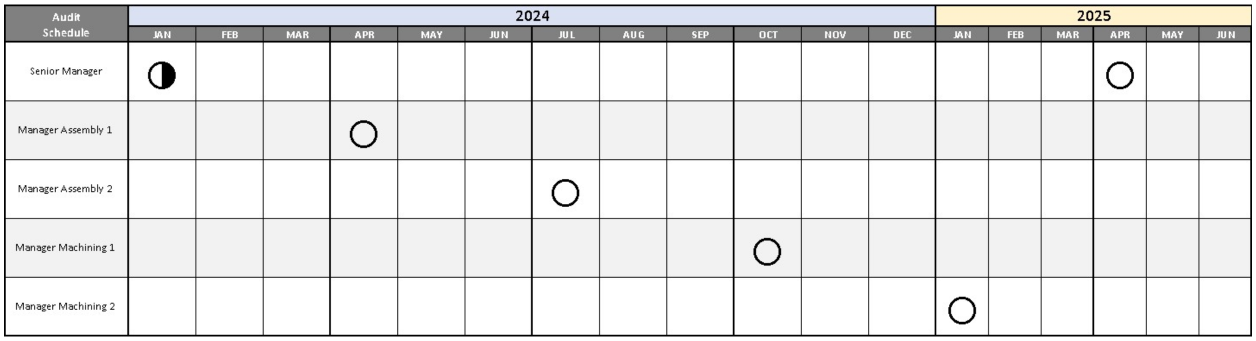
A group often shares responsibility for an area, and managing the tasks to maintain the area becomes challenging. Improving the process of maintaining an area enhances productivity, reduces stress, and ensures tasks are not forgotten. While there are many task management applications, these apps try to fit a broad scope of behavior such that they become a burden to learn. This report introduces a simple task-scheduling app concept and then provides a business proposal, system design, and system testing specifications.

Topic Approval

The primary business objective of the application is to manage audits shared by a group of auditors. Figure 1 below provides an example of an audit schedule shared by five management team members, showing how the task moves between each member and is scheduled every three months. If Manager Machining 2 is removed from the schedule, the task must be rescheduled. The audit schedule in Figure 1 only demonstrates the motion of a single task. Still, there could be hundreds of tasks and multiple groups, making a paper process confusing and cumbersome, and most task management software does not rotate through tasks in a group perpetually.

Figure 1

Audit Schedule Example



Audit Schedule	2024												2025					
	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	JAN	FEB	MAR	APR	MAY	JUN
Senior Manager	◑															○		
Manager Assembly 1				○														
Manager Assembly 2							○											
Manager Machining 1										○								
Manager Machining 2													○					

Note. Visualization of a quarterly audit scheduled across five auditors. Created using Microsoft Excel.

The task scheduling application will be a mobile app. The application will be created for Android devices and feature a card-style view of all the user's tasks. Users will create tasks using the application. Task data will be stored in a local SQLite database using a repository pattern. Users will mark a task as complete to signal the scheduling logic. The scheduling logic will automatically identify the task's next start and due date using a simple algorithm that takes the user's specification of an initial start date, the duration of the task, and the frequency at which the task will recur. This allows users to track repeatable tasks from their mobile devices easily.

Management teams often have considerable overlap in responsibility due to different shifts, key point indicator focuses, or overlapping areas of responsibility. However, a task scheduling application described above could relate to any business section with a list of tasks rotating among a group. For example, such an application could be used for household chores shared by a large family, cleaning tasks in common areas for team members, or by maintenance personnel for preventative maintenance tasks. Therefore, the task scheduling application can apply to many people in different roles despite the target audience being management teams.

Proposal

The task scheduling application will use SQLite to store task data in tables. SQLite is an embedded relational database for Android applications (McCown, 2022). Six tables have been identified to manage the tasks, schedule, responses, groups, and people. Users will populate the tables through the graphical user interface, and the information will be created, read, updated, or deleted using a repository pattern.

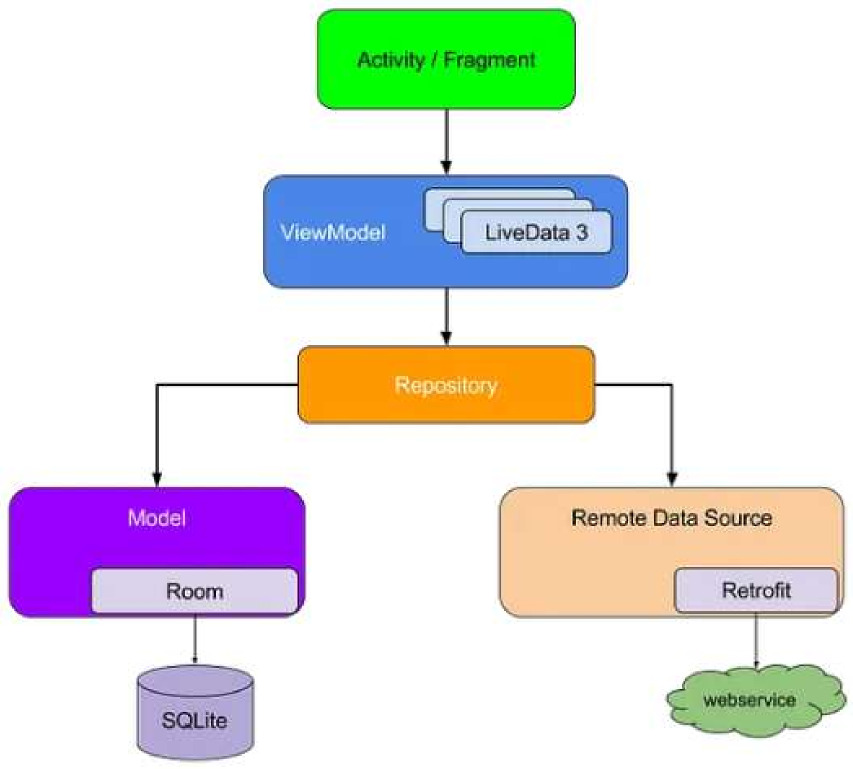
The repository pattern will move the data from the activity to the SQLite database. The repository pattern consists of four layers of abstraction from the SQLite database calls to a repository to a view model, and finally to the input/output fields in the activity or fragment that the user interacts with (see Figure 2). The repository pattern also allows for the synchronizing of data with a remote source, which opens the possibility of using a third-party database solution like MySQL and a RESTful API.

Major Software Components

The Android application will be written using the Java programming language. Android applications also use extensible markup language [XML] for the Android manifest file and app resources such as strings, colors, layouts, menus, drawables, and settings.

Figure 2

The Repository Pattern



Note. Adapted from *Repository Pattern in Android* by E. Lazuardy, 2020, Medium (https://medium.com/swlh/repository-pattern-in-android-c31d0268118c). Copyright 2020 by Ezra Lazuardy.

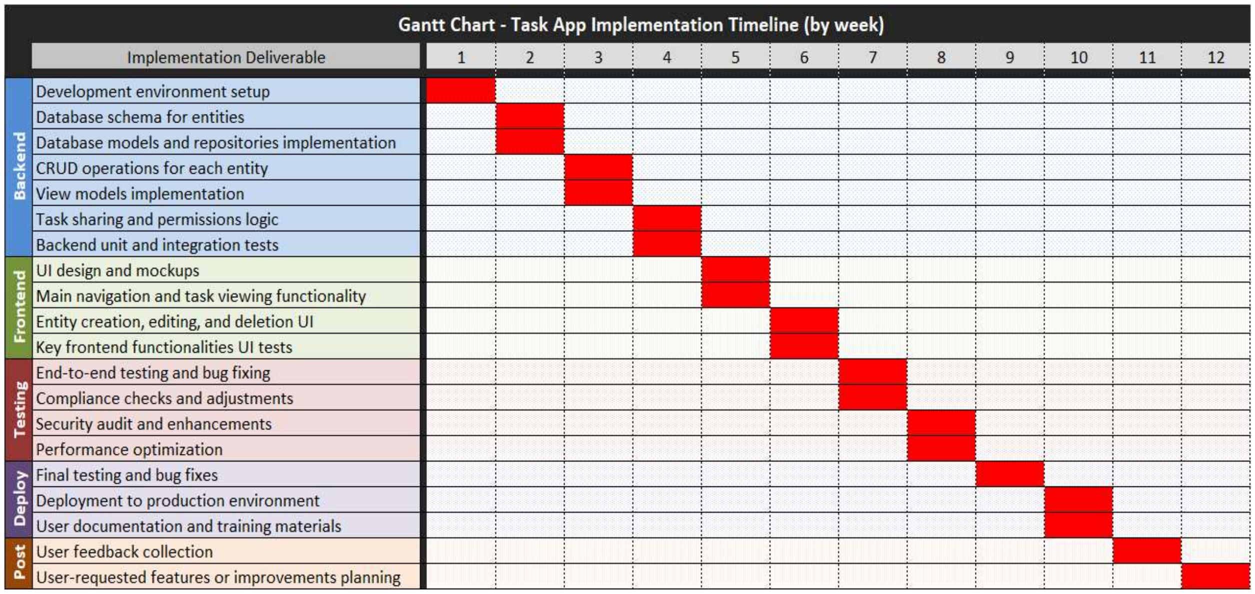
The application's user interface consists of both activities and fragments. A static class will be used to perform the scheduling logic and called by a button press event when the user completes a task. The repository pattern uses several libraries, classes, abstract classes, interfaces, and methods for database management.

Implementation Timeline

The software development life cycle [SDLC] generally consists of five phases: Analyze, design, code, test, and deploy (Unhelkar, 2018). The implementation timeline includes the code, test, and part of the deploy phases. The Gantt chart in Figure 3 visualizes the week-by-week implementation timeline over twelve weeks, with four weeks spent on the backend, two weeks spent on the front end, two weeks spent testing, two weeks for deployment, and two weeks spent analyzing customer feedback. The approach is consistent with the Waterfall development methodology, where each phase builds off the previous but is not revisited iteratively, as is seen in Agile methodologies.

Figure 3

Gantt Chart



Gantt Chart - Task App Implementation Timeline (by week)

	Implementation Deliverable	1	2	3	4	5	6	7	8	9	10	11	12
Backend	Development environment setup	🟥											
	Database schema for entities		🟥										
	Database models and repositories implementation		🟥										
	CRUD operations for each entity			🟥									
	View models implementation			🟥									
	Task sharing and permissions logic				🟥								
	Backend unit and integration tests				🟥								
Frontend	UI design and mockups					🟥							
	Main navigation and task viewing functionality					🟥							
	Entity creation, editing, and deletion UI						🟥						
	Key frontend functionalities UI tests						🟥						
Testing	End-to-end testing and bug fixing							🟥					
	Compliance checks and adjustments							🟥					
	Security audit and enhancements								🟥				
	Performance optimization								🟥				
Deploy	Final testing and bug fixes									🟥			
	Deployment to production environment										🟥		
	User documentation and training materials										🟥		
Post	User feedback collection											🟥	
	User-requested features or improvements planning												🟥

Note. Created using Microsoft Excel.

System Design

The Task Scheduling App [TSA] uses the repository pattern to move data to and from the database and user. Creating a model of the TSA solution space allows for a visual understanding of the objects, states, and components of the system. The most common models of the solution space are the class diagram and the state machine diagram. This section will provide each diagram and describe key features, then provide a table for cross-referencing.

Class Diagram

The class diagram shows the relationships between entities (Unhelkar, 2018). Constructing the class diagram began by constructing the class diagram for the Study Helper app developed by McCown (2022), which uses the same repository pattern for the underlying architecture and has a similar user interface. After reconstructing the class diagram for the Study Helper application, the names of classes, methods, and variables were adjusted to fit the context of the TSA better. Next, new classes were created, such as the Schedule entity, ScheduleDao data access object, TaskListActivity and its RecyclerView components, and GroupFragment for the sidebar from the Group Screen from the wireframe. The final result is seen in Figure 4.

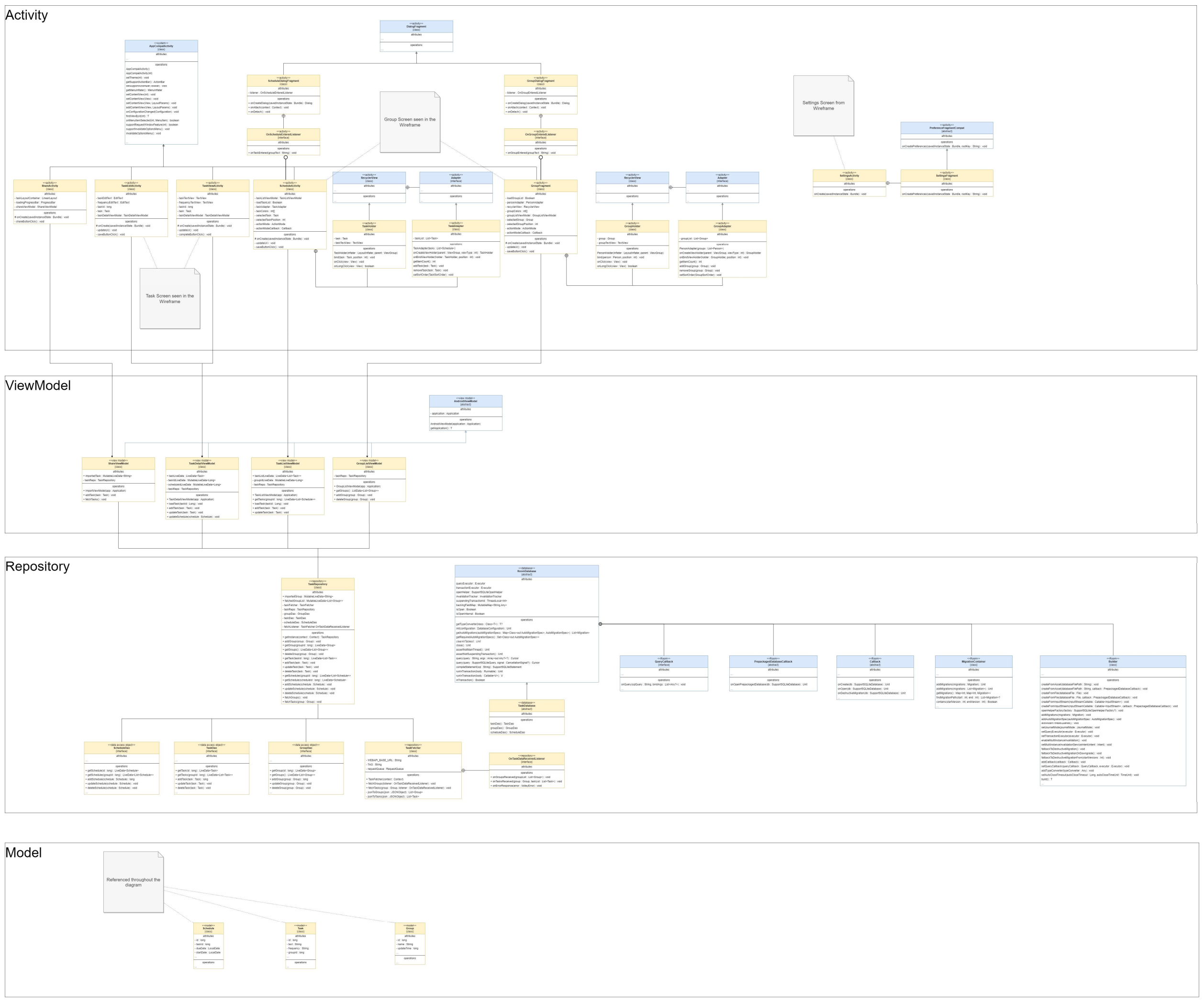
State Machine Diagram

The state machine diagram [SMD] describes how an object can change based on inputs (Unhelkar, 2018). Figure 5 shows the SMD for the Schedule object, which represents one record in the schedule table seen in the previous report’s table mapping. The purpose of the schedule table is to provide a start date, due date, response, and response date to a task object without modifying the original data. Therefore, the Schedule entity moves through seven states:

1. Scheduled: A schedule entity is created with a start date and due date,
2. Waiting: The schedule entity has a start date that is greater than the current date,
3. Ready: The schedule entity has a start date that is equal to or less than the current date,

Figure 4

Class Diagram and Architecture


Activity
ViewModel
Repository
Model
Group Screen seen in the Wireframe
Settings Screen from Wireframe
Task Screen seen in the Wireframe
Referenced throughout the diagram

Note. Created using Draw.io.

4. Overdue: The schedule entity has a due date that is less than the current date,
5. Complete: The schedule entity has been responded to by the user,
6. Rescheduled: If the schedule entity is recurring, a new schedule entity is instantiated with a new start and due date,
7. Closed: The schedule object is destroyed, but the record persists in the table.

Understanding these seven states allows the application to be developed with useful visual aids that can inform the user further, such as changing the color of the task card to red for schedule objects in the overdue state.

Cross-Reference

A previous report provided a data flow diagram, wireframe, and table mappings. Table 1 cross-references the classes found in Figure 4 with the diagrams from the previous report and lists the architectural component that the class is a member of. In cases where there isn't a clear correlation, a hyphen was used to indicate as much. The cross-reference table provides a quick reference guide for the implementation phase of the development process.

System Test

The Task Scheduling Application [TSA] is driven by user input, which leaves many points of failure. The TSA requires thorough testing to ensure users receive a reliable, high-quality product. There are several software tests available to developers that will help deliver a great user experience, such as unit tests, integration tests, and performance tests. This section will describe the testing methodologies used during development, specify how to test the TSA for each use case by identifying steps to complete the test, expected results, and inputs that could cause system failure, and discuss the performance expectations of the application.

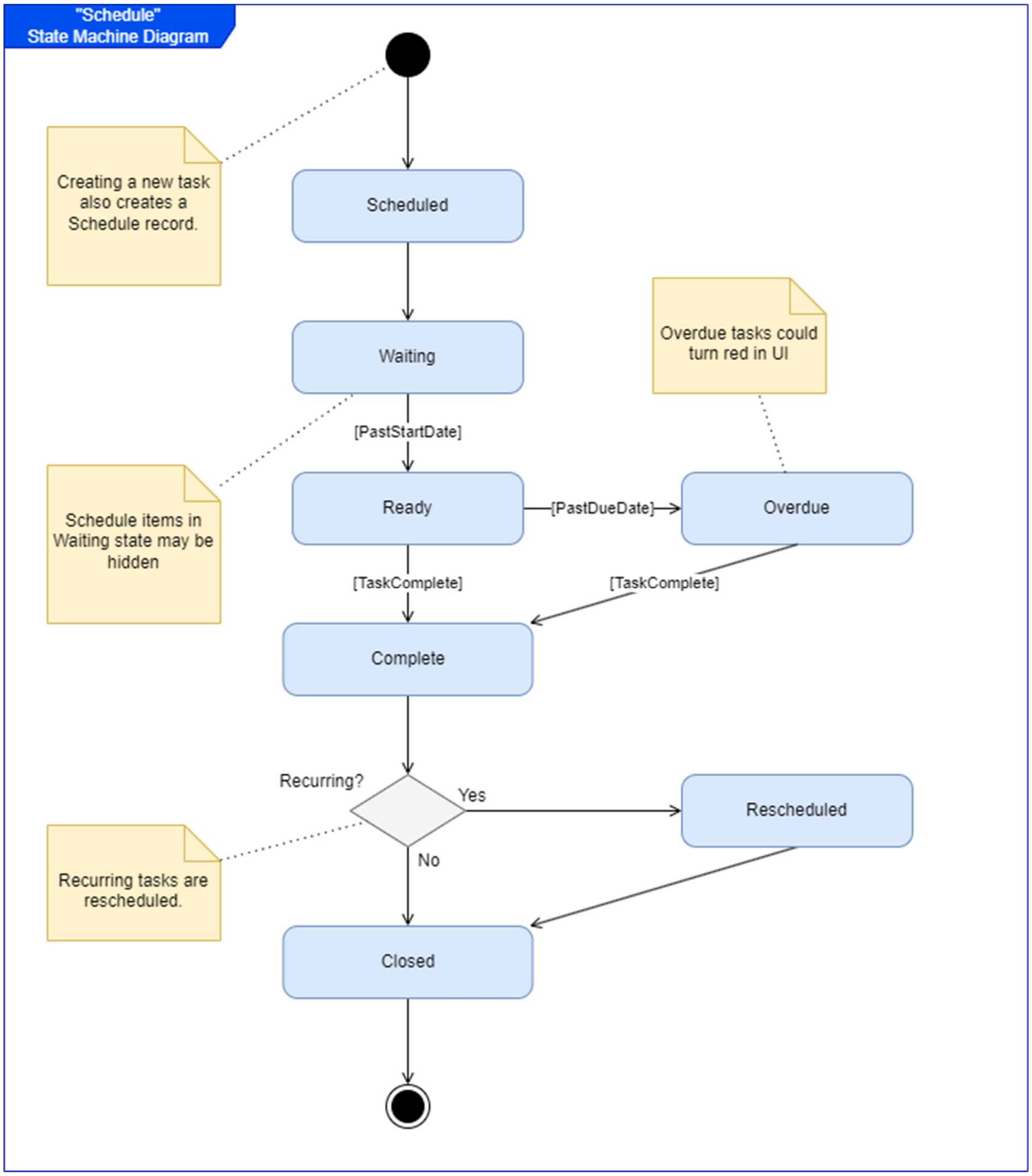
Testing phases are built into software development methodologies like agile and waterfall, typically during the implementation phases. Most tests are designed before the testing phase and performed throughout the development process. The general steps for performing tests are:

- Requirement analysis,
- Test planning,
- Test design,
- Test execution,

- Result analysis and reporting. (Johal, 2023, “What are the Steps of Software Testing?”)

Figure 5

Schedule State Machine Diagram


"Schedule" State Machine Diagram
Creating a new task also creates a Schedule record.
Scheduled
Waiting
Overdue tasks could turn red in UI
[PastStartDate]
Schedule items in Waiting state may be hidden
Ready
[PastDueDate]
Overdue
[TaskComplete]
[TaskComplete]
Complete
Recurring?
Yes
Rescheduled
No
Recurring tasks are rescheduled.
Closed

Note. The seven states of the Schedule entity. Created using Draw.io.

Table 1

Cross-Reference Table

Architecture	DFD	Class	Wireframe	Table Mapping
Model	Task System	Task	-	Task Table
Repository	Task System	TaskDao	-	-
Repository	Task System	TaskDatabase	-	-
Repository	Task System	TaskRepository	-	-
View Model	Task System	TaskDetailViewModel	-	-
Activity	Task System	TaskEditActivity	Task Activity Screen	-
Activity	Task System	TaskViewActivity	Task Activity Screen	-
Model	Group System	Group	-	Group Table
Repository	Group System	GroupDao	-	-
Repository	Group System	TaskDatabase	-	-
Repository	Group System	TaskRepository	-	-
View Model	Group System	GroupListViewModel	-	-
Activity	Group System	GroupFragment	Group Activity Screen	-
Activity	Group System	GroupHolder	Group Activity Screen	-
Activity	Group System	GroupAdapter	Group Activity Screen	-
Activity	Group System	OnGroupEnteredListener	Group Activity Screen	-
Activity	Group System	GroupDialogFragment	Group Activity Screen	-
Model	Schedule System	Schedule	-	Task-Schedule Table
Repository	Schedule System	ScheduleDao	-	-
Repository	Schedule System	TaskDatabase	-	-
Repository	Schedule System	TaskRepository	-	-
View Model	Schedule System	TaskListViewModel	-	-
Activity	Schedule System	ScheduleActivity	Group Activity Screen	-
Activity	Schedule System	TaskHolder	Group Activity Screen	-
Activity	Schedule System	TaskAdapter	Group Activity Screen	-
Activity	Schedule System	OnScheduleEnteredListener	Group Activity Screen	-
Activity	Schedule System	ScheduleDialogFragment	Group Activity Screen	-
Repository	Task Sharing System	TaskFetcher	-	-
Repository	Task Sharing System	OnTaskReceivedListener	-	-
View Model	Task Sharing System	ShareViewModel	-	-
Activity	Task Sharing System	ShareActivity	Task Activity Screen	-
Activity	-	SettingsActivity	Settings Activity Screen	-
Activity	-	SettingsFragment	Settings Activity Screen	-

Note. Each class of the class diagram is cross referenced with diagrams from previous report. Created using Microsoft Excel.

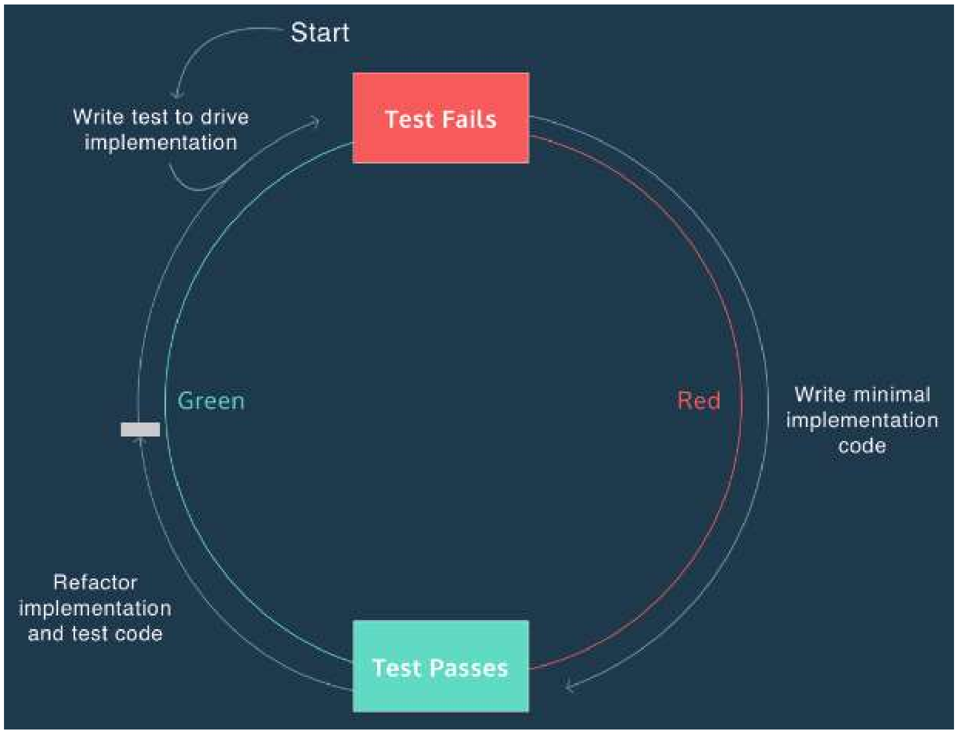
Testing Methodologies

Developing tests using the above methodology will ensure tests are thorough and lead to a high-quality product.

Test-driven development [TDD] will be used throughout the development of the TSA. In TDD, tests are written first, and then code is developed to pass the test (McCown, 2022). As seen in Figure 6, the tests and code are written using three phases: red, green, refactor (McCown, 2022). For Java and Android applications such as TSA, unit tests can be written with the JUnit framework using the TDD approach.

Figure 6

Red, Green, Refactor


Start
Write test to drive implementation
Test Fails
Green
Red
Write minimal implementation code
Refactor implementation and test code
Test Passes

Note. Three phases of test-driven development. Adapted from *Red, Green, Refactor,* by Codecademy, n.d., (https://www.codecademy.com/article/tdd-red-green-refactor). Copyright 2024 by Codecademy.

The benefits of TDD are that functions and classes are validated along the way and these small unit tests are not at risk of being ignored in favor of meeting a deadline.

Test Cases

Test cases are identified to outline the expected functionality of the application. The TSA must respond to the following five test cases:

- Create a task group,
- Create a non-recurring task,
- Edit the recurrence pattern of a task,
- Complete a recurring task,
- Share a task with a user.

This document should be updated as additional test cases are found. The test cases can be completed manually or using the Espresso testing framework (McCown, 2022). A detailed description of how each test is expected to be performed follows.

Create a Task Group

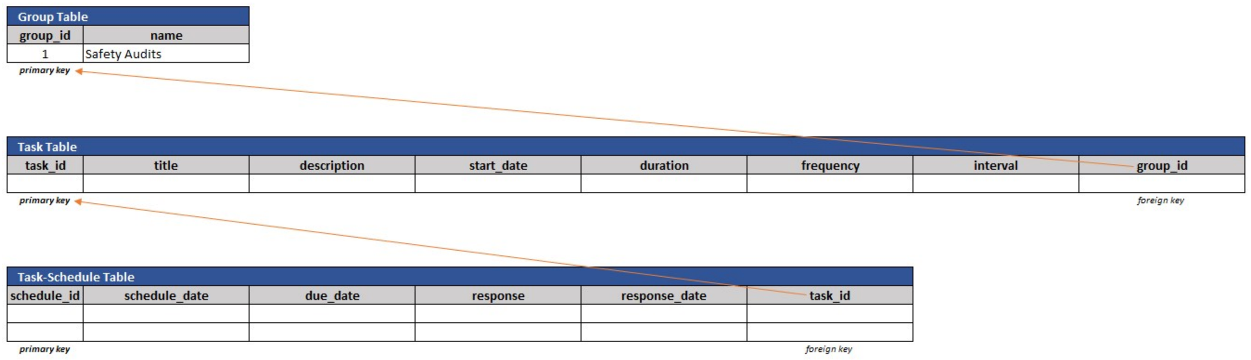
The objective of creating a task group is to verify that users can create and see the groups in a list on the user interface and confirm that the create operation functions for the Group Table. Using the input from Figure 7, complete the test as follows:

1. Expand the Group Fragment sidebar,
2. Tap the plus icon labeled "Add a new group,"
3. Enter the group name in the Group Dialog Fragment,
4. Press the save button.

The expected result is a new group listed in the sidebar with zero tasks listed in the Recycler View. Group names cannot be an empty string and cannot have special characters. The Group Table should have a new entry with the group_id and name fields containing appropriate data.

Figure 7

SQLite Tables after Creating a Task Group



Group Table

group_id	name
1	Safety Audits

Task Table

task_id	title	description	start_date	duration	frequency	interval	group_id

Task-Schedule Table

schedule_id	schedule_date	due_date	response	response_date	task_id

Note. Expected values in tables following the test case: Create a Task Group.

User input must be validated to prevent code injection and avoid database corruption. The following inputs may cause the program to fail if not validated properly:

- Special characters in the group name field,
- Empty string in the group name field.

Validation checks must be included in the onClick() function for the save button to prevent database corruption, code injection, and empty fields.

Create a Non-Recurring Task

The objective of creating a non-recurring task is to verify users can quickly add tasks to a group from the Group Activity Screen and that the create operations function for the Task Table and the Task-Schedule Table. Using the input from Figure 8, complete the test as follows:

1. Select the group that the task will belong to,
2. Tap the floating action button in the bottom right corner,
3. Enter the title of the task in the Task Dialog Fragment,
4. Enter the start date in the Task Dialog Fragment,
5. Enter the duration in the Task Dialog Fragment,

6. Press the save button.

The expected result is a new task listed in the selected group with the entered title, a new entry in the task table with the task_id, title, start_date, duration, and group_id fields populated, and a new entry in the Task-Schedule Table with the schedule_id, task_id, schedule_date, and due_date populated.

User input must be validated to prevent code injection and avoid database corruption. The following inputs may cause the program to fail if not validated properly:

- Special characters or empty string in the task title field,
- Non-numerical values in the task duration field,
- Incompatible date value in the task start_date field.

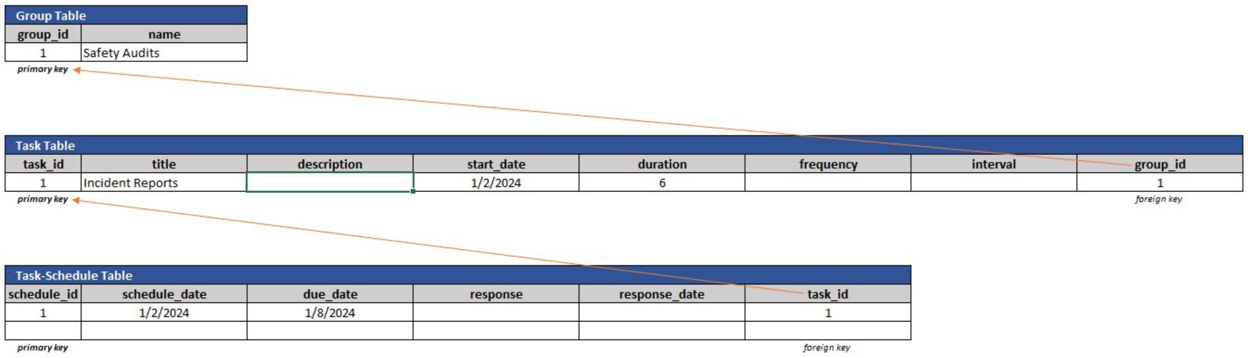
Validation checks must be included in the onClick() function for the save button to prevent database corruption, code injection, and empty fields.

Edit the Recurrence Pattern of a Task

Editing a task's recurrence pattern is intended to verify that users can edit tasks, change a task to recurring, and confirm the update operations function for the Task Table.

Figure 8

SQLite Tables after Creating a Non-Recurring Task



Group Table

group_id	name
1	Safety Audits
primary key	

Task Table

task_id	title	description	start_date	duration	frequency	interval	group_id
1	Incident Reports		1/2/2024	6			1
primary key							foreign key

Task-Schedule Table

schedule_id	schedule_date	due_date	response	response_date	task_id
1	1/2/2024	1/8/2024			1
primary key					foreign key

Note. Expected values in tables following the test case: Complete a Non-Recurring Task.

Using the input from Figure 9, complete the test as follows:

1. Select the group that the task belongs to,
2. Tap the task that needs edited from the list,
3. Wait for the Task Activity Screen to open,
4. Press the … icon to bring up a submenu,
5. Press the edit option in the submenu,
6. Enter a number in the frequency text edit box,
7. Select an interval in the drop-down box,
8. Press the save button on the bottom.

The expected result is that the non-recurring task is now recurring and the frequency and interval fields were updated for the appropriate task in the Task Table.

User input must be validated to prevent code injection and avoid database corruption. The following inputs may cause the program to fail if not validated properly:

- Special characters in the task description field,
- Non-numerical values in the task frequency field,
- Incompatible value in the task interval field,
- Duration should not exceed the quotient of the task frequency and task interval in days (e.g., a duration of 2 days should not be possible for a task recurring every 1 day).

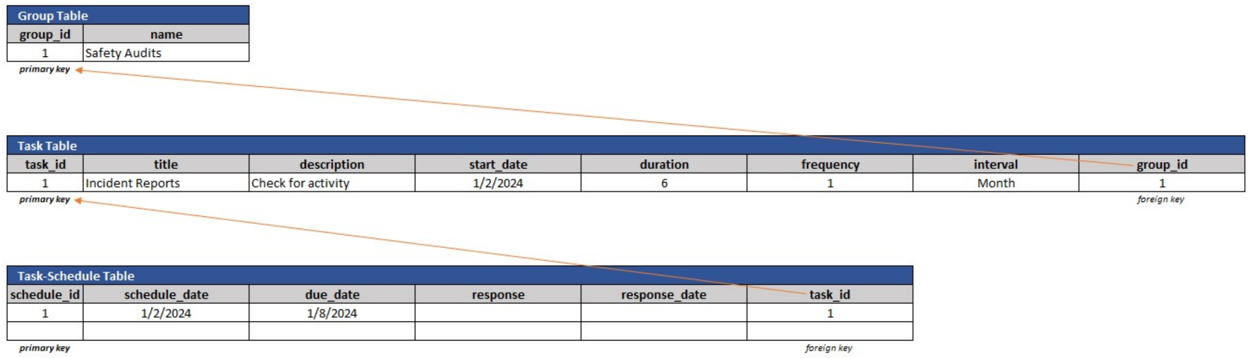
Validation checks must be included in the onClick() function for the save button to prevent database corruption and code injection.

Complete a Recurring Task

The objective of completing a recurring task is to verify users can change the state of a task from active to complete, verify the update operations function for the Task-Schedule Table, and verify the reschedule logic functions properly. The test is completed as follows:

Figure 9

SQLite Tables after Editting the Recurrence Pattern of a Task



Group Table

group_id	name
1	Safety Audits
primary key	

Task Table

task_id	title	description	start_date	duration	frequency	interval	group_id
1	Incident Reports	Check for activity	1/2/2024	6	1	Month	1
primary key							foreign key

Task-Schedule Table

schedule_id	schedule_date	due_date	response	response_date	task_id
1	1/2/2024	1/8/2024			1
primary key					foreign key

Note. Expected values in tables following the test case: Edit the Recurrence Pattern of a Task.

1. Select the group that the task belongs to,
2. Tap the task that has been completed,
3. Wait for the Task Activity Screen to open,
4. Press the Complete button.

The expected result is that the task is no longer at the top of the task list in the Group Activity Screen, the response and response_date fields are updated in the Task-Schedule Table and a new row was created in the Task-Schedule Table with the schedule_id, task_id, schedule_date, and due_date have been populated based on the task frequency and interval.

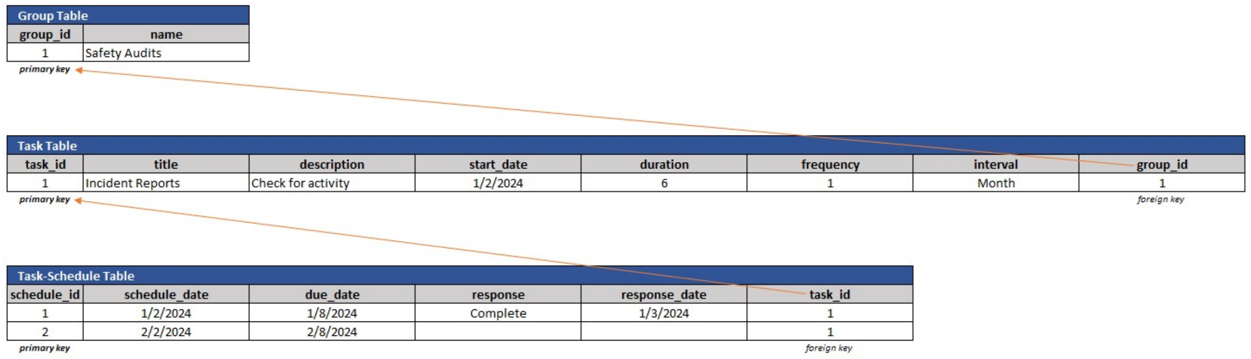
Completing a task should not cause the program to fail. However, the rescheduling logic may yield an incorrect calculation that causes the task to have incorrect values in the schedule_date and due_date. Refer to Figure 10 to confirm the reschedule logic for a once per month recurring task.

Share a Task with a User

The objective of sharing a task with a user is to confirm the task is converted to the appropriate format, the string is encrypted, the user is given the option to choose who to share the task with, and the output can be sent via SMS.

Figure 10

SQLite Tables after Complete a Recurring Task



Group Table

group_id	name
1	Safety Audits
primary key	

Task Table

task_id	title	description	start_date	duration	frequency	interval	group_id
1	Incident Reports	Check for activity	1/2/2024	6	1	Month	1
primary key							foreign key

Task-Schedule Table

schedule_id	schedule_date	due_date	response	response_date	task_id
1	1/2/2024	1/8/2024	Complete	1/3/2024	1
2	2/2/2024	2/8/2024			1
primary key					foreign key

Note. Expected values in tables following the test case: Complete a Recurring Task.

Additional test cases may be necessary to confirm reschedule logic for daily, weekly, and yearly intervals.

The test is completed as follows:

1. Select the group that the task belongs to,
2. Tap the task that is to be shared,
3. Wait for the Task Activity Screen to open,
4. Press the share button in the title bar,
5. Choose who to share the task with,
6. Press the send button.

The expected result is a file containing encrypted task data sent to the user of choice via SMS text; the receiving user should be able to tap the file, open the TSA, the task is automatically decrypted, and the user can assign the task to a group and enter a start_date; a new entry to the Task Table with the task_id, group_id, title, description, start_date, duration, frequency, and interval fields populated and a new entry to the Task-Schedule Table with the schedule_id, task_id, schedule_date, and due_date fields populated.

The sender and the recipient may have external conditions that cause the share event to fail. External conditions causing the event to fail include:

- Network errors,
- Corrupted or outdated string formats,
- Expired cryptographic keys.

The application must recover gracefully from each event. An alert must be sent informing the user that the share event failed.

Performance Expectations

Mobile applications are expected to meet several minimum performance guidelines. Four common key performance issues should be addressed:

- Scroll jank: RecyclerViews must be able to retrieve data at a 60 hz or higher rate,
- Startup latency: The TSA should be able to cold start in five seconds or less,
- Smooth transitions: New activities should not have delays or flickering,
- Power inefficiencies: Memory allocations should be optimized to prolong battery life without sacrificing app performance. (Android Open Source Project, 2024, "Key performance issues")

Performance can be measured accurately using the Macrobenchmark testing framework (Android Open Source Project, 2024). Meeting Android users' performance expectations will aid app adoption.

Conclusion

This report introduces a simple task-scheduling app concept and then provides a business proposal, system design, and system testing specifications. The project presents a task-scheduling mobile application designed to streamline the management of shared audits and tasks among groups, highlighting the challenges of existing task management systems and proposing a more straightforward, more effective solution. The business proposal outlines the app's primary objective to manage shared audits, detailing using a local SQLite database and a repository pattern for efficient data storage and retrieval, making the app suitable for various group task management scenarios. The system design section describes the application's architecture, including using Java and XML, the repository pattern for data management, and visual models like class diagrams and state machine diagrams to represent system components and task states. The implementation timeline follows the software development life cycle's code, test, and deploy phases, visualized in a Gantt chart and employing a waterfall methodology to structure the development process. System testing is crucial for ensuring a reliable user experience, using test-driven development and a range of test cases to validate app functionality, from task creation to sharing tasks with users. This project highlights the potential for such applications to streamline task scheduling in various organizational contexts, ultimately contributing to enhanced productivity and task accountability.

References

Android Open Source Project. (2024b, February 23). *Overview of measuring app performance.* Android Developers. https://developer.android.com/topic/performance/measuring-performance

Codecademy. (n.d.). *Red, green, refactor.* https://www.codecademy.com/article/tdd-red-green-refactor

Johal, D. (2023, October 4). *Software testing fundamentals: Guide to concepts and processes.* Testsigma. https://testsigma.com/blog/fundamentals-of-software-testing/

Keeling, M. (2017). *Design it! From programmer to software architect* (1st ed.). Pragmatic Bookshelf.

Lazuardy, E. (2020, August 1). *Repository pattern in Android.* Medium. https://medium.com/swlh/repository-pattern-in-android-c31d0268118c

McCown, F. (2022). *Mobile app development with Android and Java*. Wiley: zyBooks.

Unhelkar, B. (2018). *Software engineering with UML* (1st ed.). CRC Press.