Capstone Proposal Project

Name: Routines

Student Name: Taylor Chesbro

Table of Contents

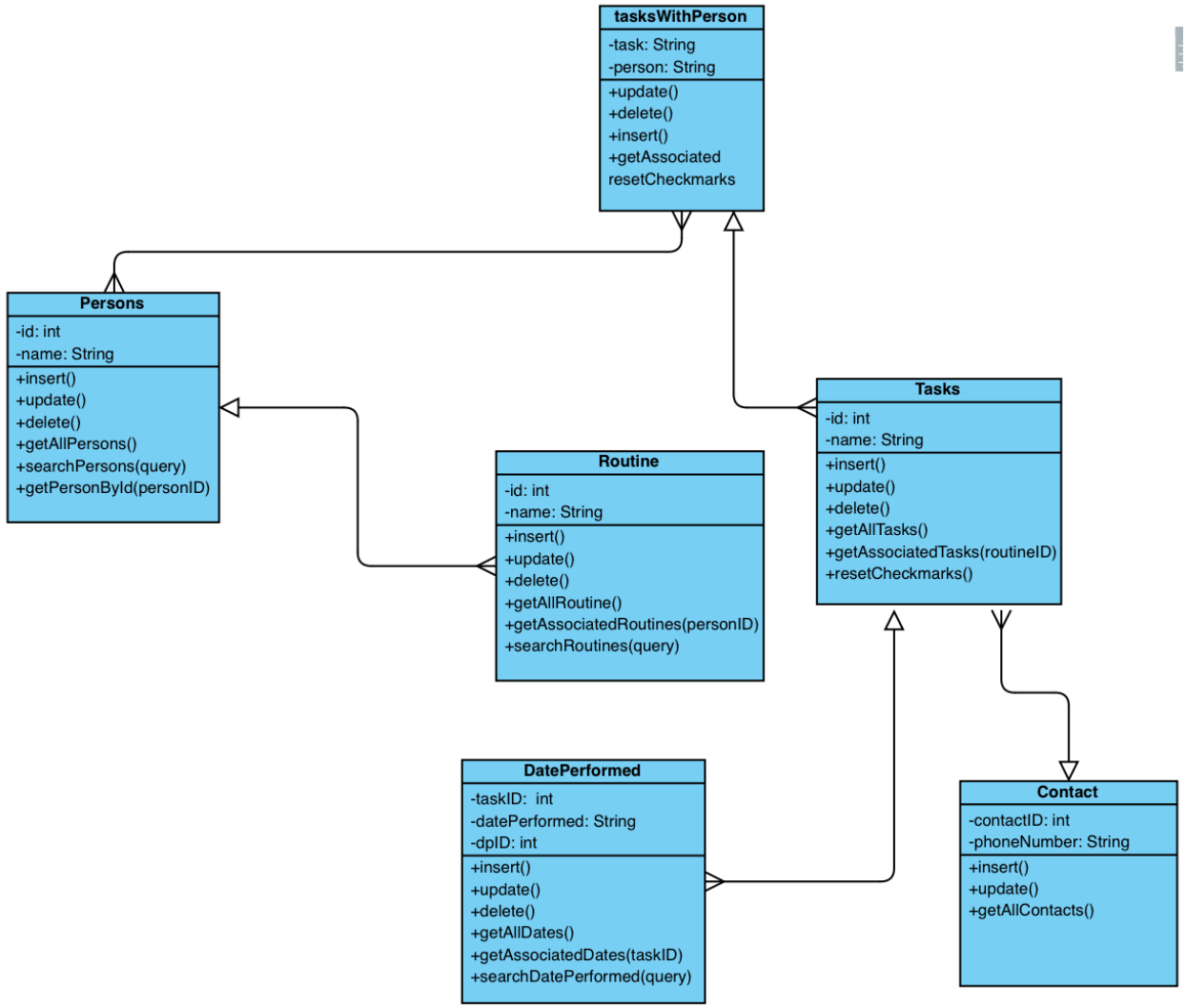
Application Design and Testing

Class Design

The provided diagram is a class diagram that outlines a structured application that is based on creating and maintaining routines. The application is built with Android studio using Java language. Since Java is an object-oriented language, the classes are distinctly shown in the diagram.

The application takes a person class that then can have many routines associated to the person. From there, there can be many tasks associated to each routine. There is also a tasksWithPerson class that is used for searching which person is associated to what task or list all the tasks a person has no matter which routine. This search can help the user remember who is associated to a certain task if they forgot but also can create a report of all the tasks a person is responsible for. The datePerformed class is a many to one relationship to a task in the Task class. The contact class then saves contact information that is a one to many relationship with the Task class, as all task will use the one contact. Each class has methods that update, insert and delete. Most also have methods to query, getAll and getAssociated classes.

The application will store the data using Room within android studio. Each class has adapters (not shown in the diagram) that help RecyclerViews load from the database upon the pages. There is also a contact class that holds contact information that is used by Tasks to contact the saved contact automatically.



UI Design

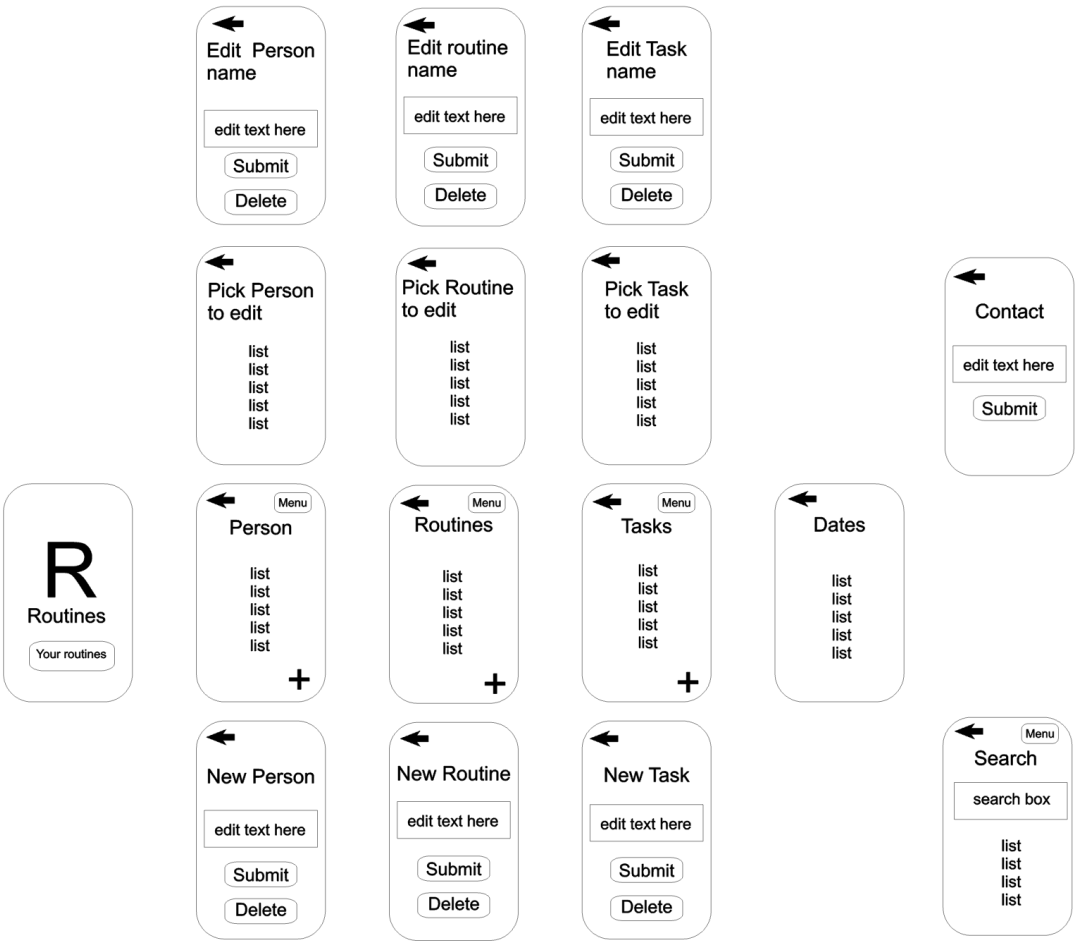
This user interface design of the Routines application outlines a simple and intuitive navigation system for managing task assigned to routines that are assigned to a person along with the dates completed. It features multiple screens, each dedicated to specific operations.

The user interface features easily contrasted colors throughout the application and easy to read fonts. The key colors are: khaki, orange, blue, green and white. The colors and basic design

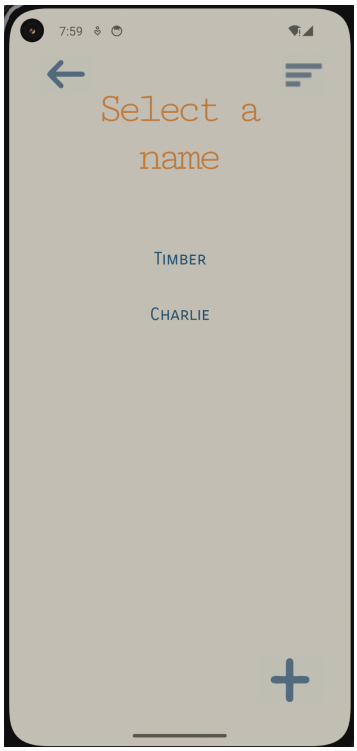
were chosen to create an ease of use for all that use it - knowing that a variety of ages will be using it. There are choices such as editing, choosing, deleting or creating entries for persons, routines, and tasks. It also includes a search feature and contact info screen .

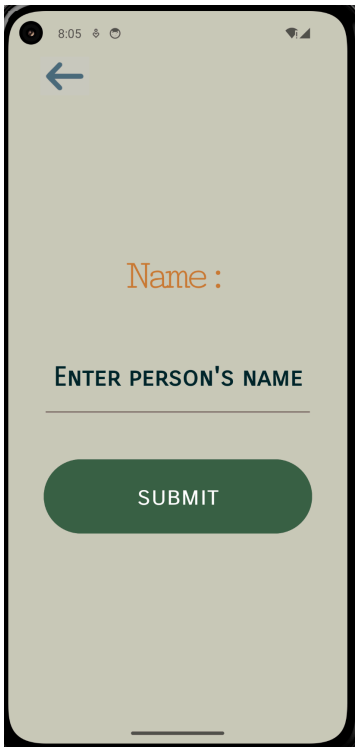
Key components include lists for selecting items to edit, text fields for making modifications or adding new entries, and buttons for submitting or deleting data. A menu option gives an easy way to return home and there is also a back button ensuring seamless navigation. The minimalist design focuses on functionality, making it ideal for a productivity application that prioritizes clarity and ease of use.

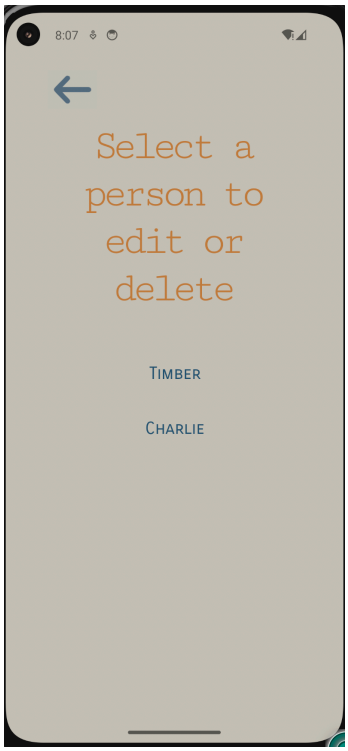
The application also features many toast notifications that appear on screen while using the app. These toast notifications allow the user to understand text character limits (prevent SQL injections) and confirm their changes. There is also a toast notification when a checkbox is marked that sends an SMS of completion of the task. The contact for this is saved and updated from the menu option.

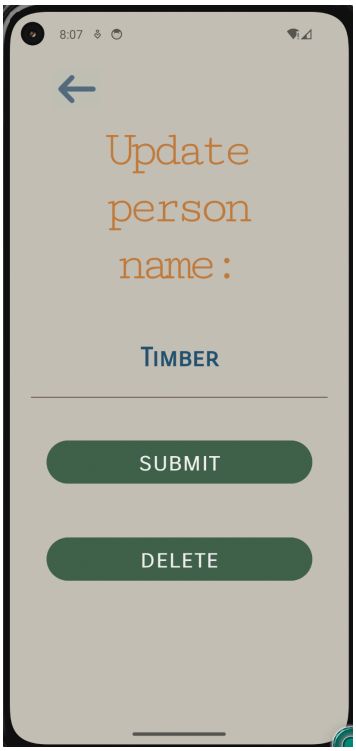


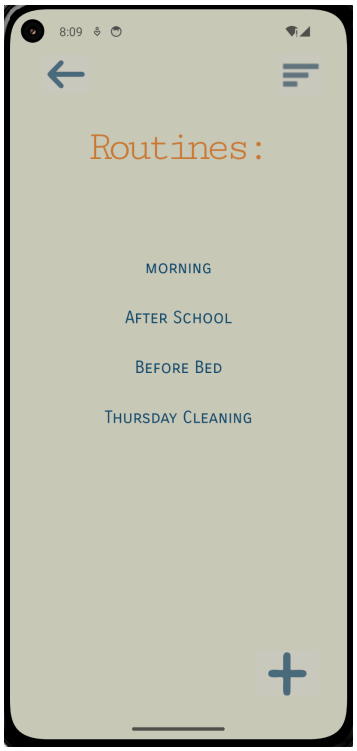
Low fidelity image

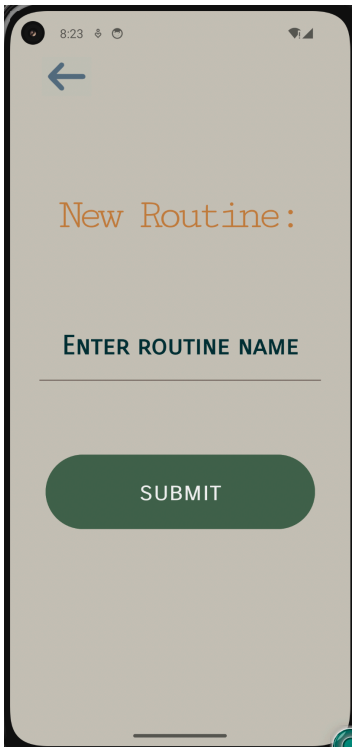

7:59
Select a name
TIMBER
CHARLIE

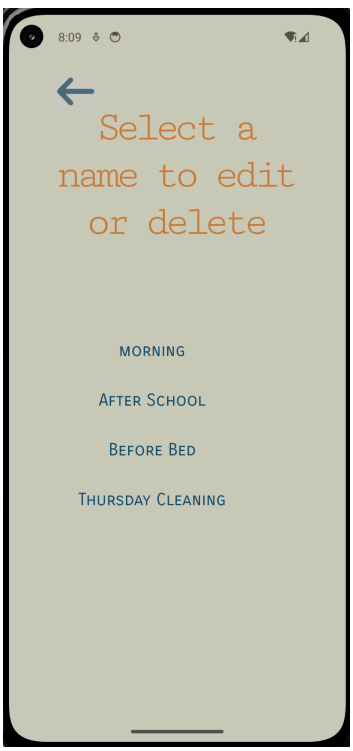

8:05
Name:
ENTER PERSON'S NAME
SUBMIT

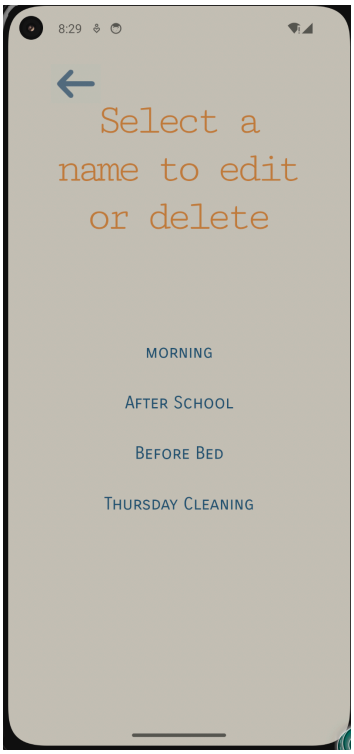

8:07
Select a person to edit or delete
TIMBER
CHARLIE

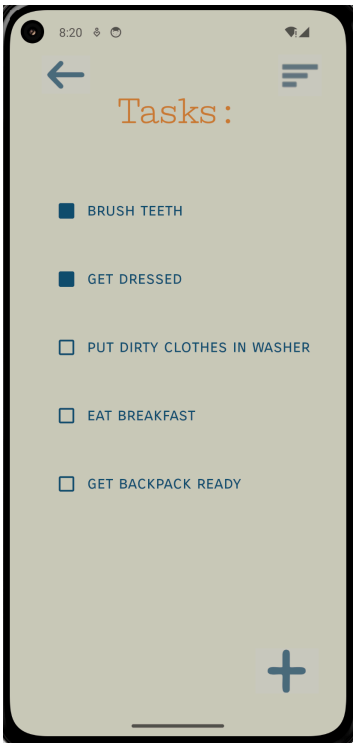

8:07
Update person name:
TIMBER
SUBMIT
DELETE

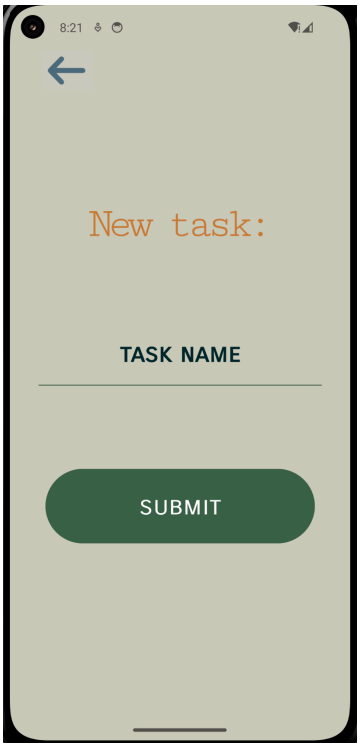

8:09
Routines:
MORNING
AFTER SCHOOL
BEFORE BED
THURSDAY CLEANING


8:23
New Routine:
ENTER ROUTINE NAME
SUBMIT

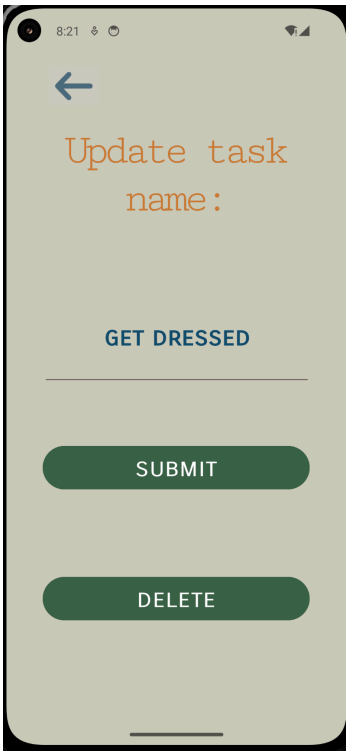

8:09
Select a name to edit or delete
MORNING
AFTER SCHOOL
BEFORE BED
THURSDAY CLEANING

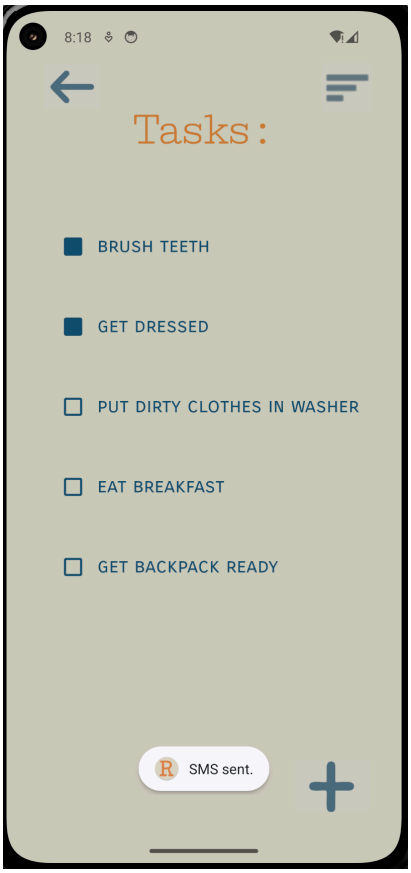

8:29
Select a name to edit or delete
MORNING
AFTER SCHOOL
BEFORE BED
THURSDAY CLEANING

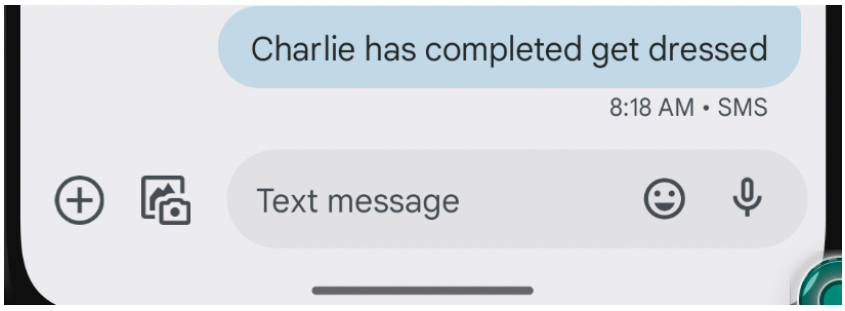

8:20
Tasks:
BRUSH TEETH
GET DRESSED
PUT DIRTY CLOTHES IN WASHER
EAT BREAKFAST
GET BACKPACK READY


8:21
New task:
TASK NAME
SUBMIT


8:20
Select a task to edit or delete
BRUSH TEETH
GET DRESSED
PUT DIRTY CLOTHES IN WASHER
EAT BREAKFAST
GET BACKPACK READY


8:21
Update task name:
GET DRESSED
SUBMIT
DELETE


8:18
Tasks:
BRUSH TEETH
GET DRESSED
PUT DIRTY CLOTHES IN WASHER
EAT BREAKFAST
GET BACKPACK READY
SMS sent.


Charlie has completed get dressed
8:18 AM • SMS
Text message

Unit Test Plan

Introduction

Purpose

The purpose of the unit test plan is to ensure the Routines Application codebase is both accurate and reliable. A comprehensive testing strategy was implemented to thoroughly verify the functionality of key components. This approach ensured that the application operates as intended, while maintaining the highest quality by identifying and resolving any defects.

Overview

The unit test plan is an important part of the development process for the Routines Application. The testing methods for this application are tailored to testing the significant features that make the application unique and important.

The tests encompassed a variety of scenarios such as getting and setting the tasks and persons, along with getting the task and persons in relation to forming the person - task relationship that is shown in the search feature. This comprehensive approach ensured that the application's search feature pulled the accurate information and worked as expected both individually and in combination.

- Task - Person relationships for the search feature: To ensure the TaskWithPerson entity works as expected, comprehensive unit tests were implemented. These tests validated the functionality of getter and setter methods, ensuring they correctly handle the task and person properties. Additionally, the test verified the getSearchableName method, confirming it concatenates the task name and person name appropriately.

Test Plan

Items

- Development Environment:
    - Android Studio installed with the latest stable version.
    - Java Development Kit (JDK) configured properly for the Android project.
- Application Source Code:
    - The source code of the Android application cloned from the repository.
    - Test scripts included within the project structure
- Test Framework:
    - JUnit: The default testing framework for unit tests in Android.
- Assertion Library:
    - Built-in assertions provided by JUnit
- Build Tool:
    - Gradle: integrates with testing tools and runs tests as part of the build process.

Features

- Search for Task - Person relationships: The method that connects the people to task for searching purposes visually
- Getters and Setters - getting and setting Tasks and Persons is a very important part of the application, the test verified they work properly

Deliverables

•Test Scripts: The test scrips are a collection of source codebase

•Test Results: A report that gives the results of the test and tells what passed and failed along with the messages

```java
package com.example.routines;

import static org.junit.Assert.assertEquals;

import com.example.routines.entities.Person;
import com.example.routines.entities.TaskWithPerson;
import com.example.routines.entities.Tasks;

import org.junit.Before;
import org.junit.Test;

public class TaskWithPersonTest {

    12 usages
    private TaskWithPerson taskWithPerson;
    7 usages
    private Tasks task;
    7 usages
    private Person person;

    @Before
    public void setUp() {
        taskWithPerson = new TaskWithPerson();
        task = new Tasks();
        person = new Person();

        task.setTaskName("Sample Task");
        person.setPersonName("John Doe");
    }

    @Test
    public void testGetTask() {
        taskWithPerson.setTask(task);
        assertEquals(task, taskWithPerson.getTask());
    }
```

```
32      }
33
34      @Test
35      public void testSetTask() {
36          taskWithPerson.setTask(task);
37          assertEquals(task, taskWithPerson.getTask());
38      }
39
40      @Test
41      public void testGetPerson() {
42          taskWithPerson.setPerson(person);
43          assertEquals(person, taskWithPerson.getPerson());
44      }
45
46      @Test
47      public void testSetPerson() {
48          taskWithPerson.setPerson(person);
49          assertEquals(person, taskWithPerson.getPerson());
50      }
51
52      @Test
53      public void testGetSearchableName() {
54          taskWithPerson.setTask(task);
55          taskWithPerson.setPerson(person);
56          assertEquals( expected: "Sample Task - John Doe", taskWithPerson.getSearchableName());
57      }
58  }
```

Tasks

- **Set up the development environment**
    - JUnit
    - Android Studio with a properly configured Gradle environment
    - JDK - Java development Kit installed and set up on the system
    - Ensure that the Android SDK is installed
    - Import any necessary dependencies

- **Ensure Test Scripts are in place**
    - Tests for the setTask() and getTask() methods
    - Tests for setPerson() and getPerson() methods
    - Test for getSearchableName() method with intent it returns concatenation of task and person names
- **Execute the Test Scripts**
    - The tests are executed using JUnit, with each method annotated by the @Test annotation:
        - Test Method 1: testGetTask() — Verifies that the getTask() method correctly retrieves the set Task.
        - Test Method 2: testSetTask() — Verifies that the setTask() method correctly assigns a Task with the TaskWithPerson instance.
        - Test Method 3: testGetPerson() — Verifies that the getPerson() method correctly retrieves the set Person.
        - Test Method 4: testSetPerson() — Verifies that the setPerson() method correctly assigns a Person with the TaskWithPerson instance.
        - Test Method 5: testGetSearchableName() — Verifies that the getSearchableName() method concatenates the task and person names properly.
    - Run the tests directly from Android Studio by right-clicking the test class or individual test method and selecting Run.
- **Monitor and analyze test results:**

- Once the tests are executed, monitor the results in the JUnit Console or Test Results tab in Android Studio.
- You should see test results showing that each test passed, and any failed tests will display an error message with details about the failure.
- If any test fails, you will be provided with the relevant failure message and stack trace to help with debugging.

- **Review the Test Results**
  - The completed test results should show that all assertions passed:
  - taskWithPerson.getTask() returns the correct Task.
  - taskWithPerson.getPerson() returns the correct Person.
  - taskWithPerson.getSearchableName() returns "Sample Task - John Doe".

```java
package com.example.routines;

import static org.junit.Assert.assertEquals;

import com.example.routines.entities.Person;
import com.example.routines.entities.TaskWithPerson;
import com.example.routines.entities.Tasks;

import org.junit.Before;
import org.junit.Test;

public class TaskWithPersonTest {

    12 usages
    private TaskWithPerson taskWithPerson;
    7 usages
    private Tasks task;
    7 usages
    private Person person;

    @Before
    public void setUp() {
        taskWithPerson = new TaskWithPerson();
        task = new Tasks();
        person = new Person();

        task.setTaskName("Sample Task");
        person.setPersonName("John Doe");
    }

    @Test
    public void testGetTask() {
        taskWithPerson.setTask(task);
        assertEquals(task, taskWithPerson.getTask());
    }
```

```
    }

    @Test
    public void testSetTask() {
        taskWithPerson.setTask(task);
        assertEquals(task, taskWithPerson.getTask());
    }

    @Test
    public void testGetPerson() {
        taskWithPerson.setPerson(person);
        assertEquals(person, taskWithPerson.getPerson());
    }

    @Test
    public void testSetPerson() {
        taskWithPerson.setPerson(person);
        assertEquals(person, taskWithPerson.getPerson());
    }

    @Test
    public void testGetSearchableName() {
        taskWithPerson.setTask(task);
        taskWithPerson.setPerson(person);
        assertEquals( expected: "Sample Task - John Doe", taskWithPerson.getSearchableName());
    }
}
```

Needs

- **Software Requirements:** The development environment must have the necessary dependencies installed, including JDK, Android Studio, and Gradle. Ensuring that the correct versions of the JDK and Android Studio are used is critical for compatibility with Android-specific libraries and the build system.
- **Testing Tools and Libraries:** JUnit (preferably JUnit 4 or 5) should be integrated as the testing framework. AndroidJUnitRunner may also be required for Android-specific testing. Additionally, any assertion libraries that are used in the codebase need to be correctly configured to ensure proper unit test execution.

- **Access to Source Code and Test Scripts:** Access to the project's source code and test scripts are required to review and ensure that unit tests are comprehensive and correctly implemented. The test scripts are located in the easy to find files labeled test. A version control system, like Git, should be used to manage code changes, track modifications to the test scripts, and ensure that tests remain up-to-date as the application evolves.
- **Test Data:** Predefined test data needs to be available for the unit tests. This includes sample data for objects like Task and Person, which are used in the TaskWithPersonTest. Test data for this is hardcoded in the test setup.

Pass/Fail Criteria

Test: testGetTask()

- Pass: The getTask() method correctly retrieves the Task object that was previously set using setTask().
- Fail: The getTask() method does not return the correct Task object, or it returns null.

Test: testSetTask()

- Pass: The setTask() method correctly assigns the Task object to the TaskWithPerson instance, and getTask() returns the same object.
- Fail: The setTask() method does not properly assign the Task object, or getTask() returns a different object than the one set.

Test: testGetPerson()

- Pass: The getPerson() method correctly retrieves the Person object that was set using setPerson().
- Fail: The getPerson() method does not return the correct Person object, or it returns null.

Test: testSetPerson()

- Pass: The setPerson() method correctly assigns the Person object to the TaskWithPerson instance, and getPerson() returns the same object.
- Fail: The setPerson() method does not properly assign the Person object, or getPerson() returns a different object than the one set.

Test: testGetSearchableName()

- Pass: The getSearchableName() method correctly concatenates the Task and Person names (e.g., "Sample Task - John Doe").
- Fail: The getSearchableName() method returns an incorrect or empty string, or fails to concatenate the Task and Person names correctly.

Summary of Pass/Fail Results

- A tests **passes** if:
  - The getTask() method returns the correct Task object.
  - The setTask() method correctly assigns the Task object.
  - The getPerson() method returns the correct Person object.
  - The setPerson() method correctly assigns the Person object.

        - The getSearchableName() method concatenates the Task and Person names correctly.

- A test **fails** if:
    - If any of the above conditions are not met, the respective test will fail, and the issue will need to be investigated

*If a test fails during the testing process, the following remediation strategies and documentation practices are applied to resolve the issue:

- Identify the Root Cause: The first step is to thoroughly investigate the failed test by reviewing error messages, logs, and any relevant stack traces that might provide insights into the failure. For example, if a test such as testGetTask() fails, examining the test method to ensure that the Task object is being set and retrieved correctly. Then checking for potential issues in the TaskWithPerson class such as incorrect object assignments or null pointer exceptions.
- Document the Issue: Once the root cause is identified, a bug report or issue ticket is created to formally document the problem. The report includes details such as:
    - The name of the test that failed (e.g., testGetTask()).
    - The specific error message or exception encountered during the test (e.g., "Expected Task object, but got null").
    - Steps to reproduce the failure, including any specific conditions or inputs that caused the issue.

- Any relevant system or environment information that may impact the test, such as changes in dependencies or codebase.
- A suggested course of action or fix, such as revising the setter methods, ensuring correct object initialization, or debugging related logic.

Specifications

Here are some examples of the test code screenshots from the Routines source codebase.

```
    @Test
    public void testGetSearchableName() {
        taskWithPerson.setTask(task);
        taskWithPerson.setPerson(person);
        assertEquals( expected: "Sample Task - John Doe", taskWithPerson.getSearchableName());
    }
}
```

Procedures

- Test Case Preparation: The critical functionalities of the TaskWithPerson class, such as setting and getting Task and Person objects, and verifying the correct concatenation of their names, are the key areas for testing. Test cases were written to cover these scenarios, ensuring that both the setTask(), getTask(), setPerson(), getPerson(), and getSearchableName() methods are properly functioning. The expected outcomes and pass/fail criteria for each test case were clearly defined, such as ensuring the correct Task and Person objects are returned and the getSearchableName() method correctly concatenates their names.

- Test Environment Setup: The test environment was set up within Android Studio along with configuring the necessary dependencies such as JUnit. The Android SDK and Gradle build system were verified to be properly installed and configured. Testing libraries like JUnit were correctly integrated, ensuring that unit tests could run without issues in the Android Studio terminal.
- Test Execution: The test suite was executed using JUnit through Android Studio's built-in test runner. All test cases were run, and the results were recorded. During the execution of the tests, each test was analyzed to determine if the expected outcomes were met. If any test failed, detailed error logs and stack traces were reviewed to identify the cause of the failure, such as issues in object assignment or string concatenation logic.
- Test Results Review and Documentation: The test results were documented, including pass/fail outcomes for each test case. All the tests passed during this case but protocol would require any identified issues, such as failures in the getTask() or getSearchableName() methods be noted, and any changes made to the application code or test cases would be documented. The test plan and individual test cases were updated to reflect the final outcomes of the tests, ensuring that the process was fully documented.

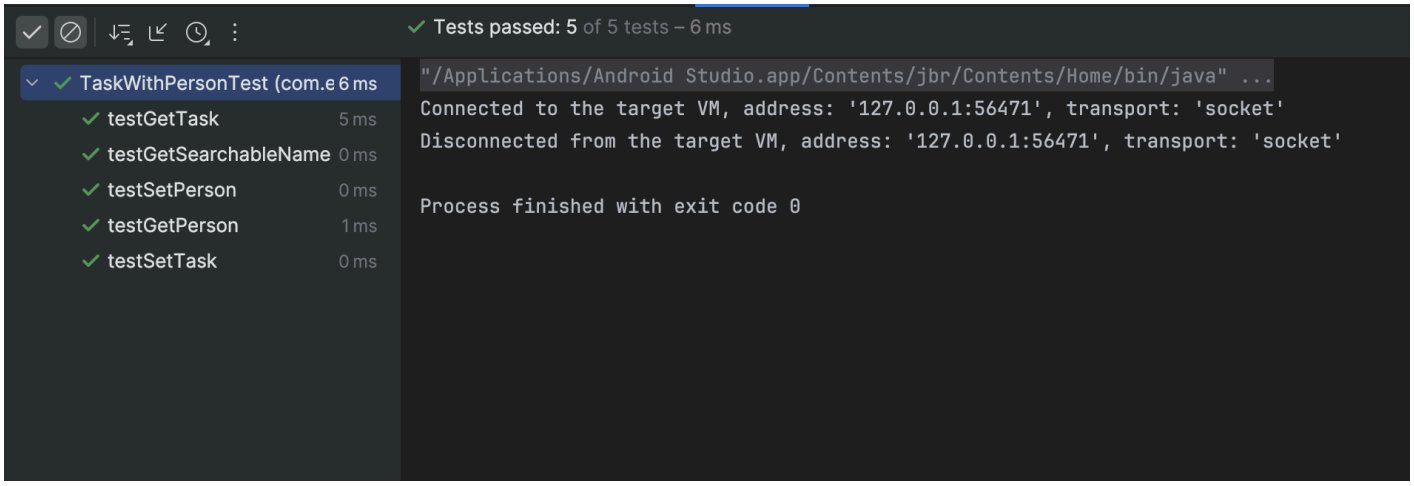
Results

- Task and Person Assignment: The test passed when the getTask() method correctly returned the assigned Task and the setTask() method successfully assigned a Task object to the TaskWithPerson instance. Similarly, when the getPerson() method

successfully returned the correct Person and setPerson() method passed when it correctly assigned a Person object to the TaskWithPerson instance.

- Searchable Name Generation: The test passed when the getSearchableName() method correctly concatenated the task name and person name (e.g., "Sample Task - John Doe"). This ensured that the method provided the expected output based on the associated task and person.
- Edge Case Handling: When testing edge cases, such as assigning null values or empty strings, the tests passed when the appropriate behavior was observed, such as null checks or validation of empty data. This confirmed that the TaskWithPerson class could handle various scenarios without failure.

These test results provided assurance that the core functionality of the TaskWithPerson class was working as expected, ensuring the reliability and correctness of the application. Results screenshot: all tests passed.



```java
package com.example.routines;

import static org.junit.Assert.assertEquals;

import com.example.routines.entities.Person;
import com.example.routines.entities.TaskWithPerson;
import com.example.routines.entities.Tasks;

import org.junit.Before;
import org.junit.Test;

public class TaskWithPersonTest {

    12 usages
    private TaskWithPerson taskWithPerson;
    7 usages
    private Tasks task;
    7 usages
    private Person person;

    @Before
    public void setUp() {
        taskWithPerson = new TaskWithPerson();
        task = new Tasks();
        person = new Person();

        task.setTaskName("Sample Task");
        person.setPersonName("John Doe");
    }
```

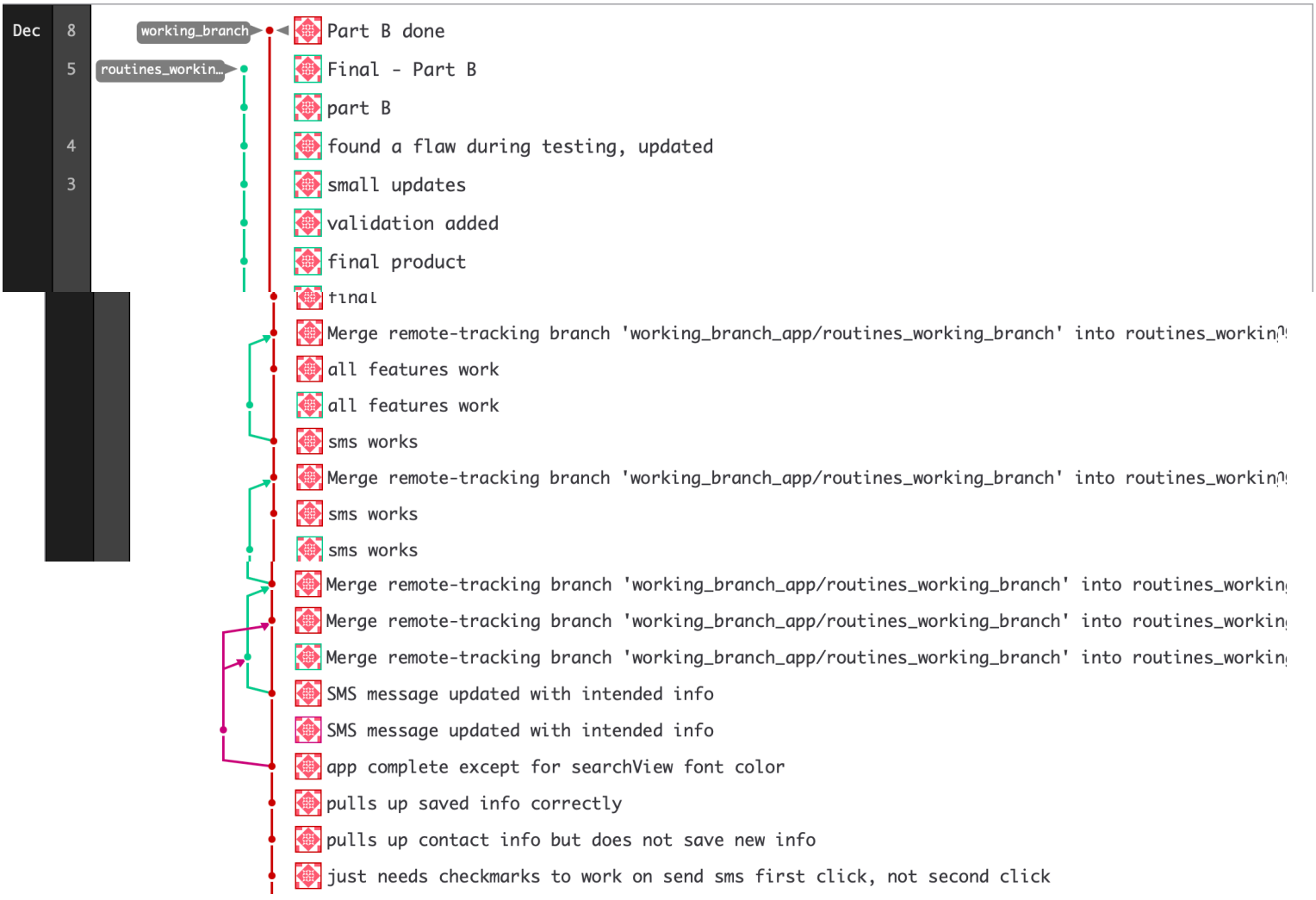
```java
    @Test
    public void testGetTask() {
        taskWithPerson.setTask(task);
        assertEquals(task, taskWithPerson.getTask());
    }

    @Test
    public void testSetTask() {
        taskWithPerson.setTask(task);
        assertEquals(task, taskWithPerson.getTask());
    }

    @Test
    public void testGetPerson() {
        taskWithPerson.setPerson(person);
        assertEquals(person, taskWithPerson.getPerson());
    }

    @Test
    public void testSetPerson() {
        taskWithPerson.setPerson(person);
        assertEquals(person, taskWithPerson.getPerson());
    }
```

Application made for Android Devices

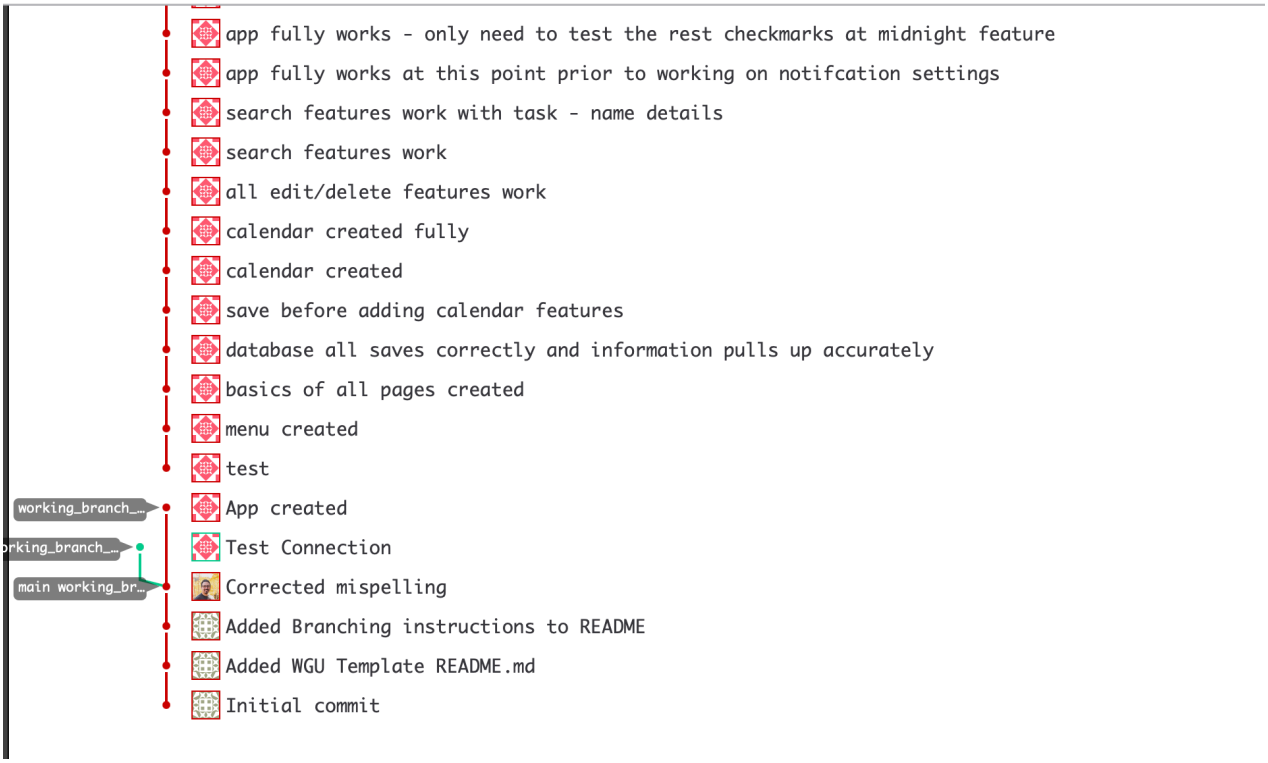
Will be hosted via Google Play Store

GitLab Repository & Branch History

GitLab Repository Link: https://gitlab.com/wgu-gitlab-environment/student-repos/tchesbr/d424-software-engineering-capstone.git



.


app fully works - only need to test the rest checkmarks at midnight feature
app fully works at this point prior to working on notifcation settings
search features work with task - name details
search features work
all edit/delete features work
calendar created fully
calendar created
save before adding calendar features
database all saves correctly and information pulls up accurately
basics of all pages created
menu created
test
working_branch_...
App created
rking_branch_...
Test Connection
main working_br...
Corrected mispelling
Added Branching instructions to README
Added WGU Template README.md
Initial commit

User Guide for Initial Setup & Running the Application

Introduction

This User Guide provides a walk through of the initial set up of the Routines application and how to get it up and running for maintenance and testing purposes.

Clone the project from Gitlab, install packages and other dependencies

1. Git Clone Command

2. **Git clone** https://gitlab.com/wgu-gitlab-environment/student-repos/tchesbr/d424-software-engineering-capstone.git

    ***Note**: This step clones the repository to your local machine. Ensure you have Git installed and configured properly.

3. Opening the Project in Android Studio

    a. Open Android Studio

    b. Click File

    c. Click New…

    d. Click Project from Version Control..

    e. A screen will pop up asking for a URL

    f. Paste the clone URL into the URL spot

4. Trusting the Project

    g. A new project will appear, click trusted if a screen pops up asking

        * **Explanation**: Android Studio may prompt you to trust the project, if it's from an external source. Trusting the project allows Android Studio to execute it.

5. Branch Management

h. Click the branch that says main, go to “working_branch” under remote and click checkout

6. Handling Uncommitted Files

If an error pops up saying uncommitted files are preventing checkout:

i. Click terminal

j. Type:

k. git add .

l. git stash

***Caution:** This step stashes the untracked files. While the files most likely wouldn’t affect the workflow if committed. Stashing them is the more secure option to keep project integrity.

m. Now go back to the branches tree

n. Find “working_branch” under remote

o. Click Checkout

p. Now the files should be loaded

7. Invalidating Caches

q. Click File -> Invalidate Caches

r. Accept and let it restart

8. Building the Project

s. Once loaded, go to terminal

t. Type ./gradlew build

*** Note:** Ensure that the gradlew script has execute permissions. If not, you
can make it executable with:
* chmod +x gradlew

u. It will now show as an app (You will see the android icon by run and to confirm, the drop-down menu on the top left that says project will show an android option)

9. Cleaning and Rebuilding the Project

v. Click Build -> Clean Project

w. Click Build -> Rebuild Project

10. Running the Emulator

x. An emulator will appear with the project

y. If the emulator does not appear, click the run button

* Potential Issue: If the emulator doesn't start automatically:
- Ensure that you have an emulator configured in AVD Manager.
- Verify that your system meets the requirements to run the emulator.
- Sometimes, restarting Android Studio or your computer can resolve emulator startup issues.

Additional Suggestions

1. Prerequisites:

a. Java Development Kit (JDK): Ensure that the appropriate version of JDK is

installed and configured.

b. Android SDK: Verify that the required SDK versions for the project are installed via the SDK Manager.

2. Dependencies:

d. After cloning the repository, Android Studio should automatically handle dependency resolution. However, if you encounter issues, try:

*Syncing Gradle: Click on File ->Sync Project with Gradle Files.

*If any font is red, try hovering over it. If it asks to import, click import to import that dependency

3. Running on Physical Devices:

e. While emulators are convenient, testing on physical devices can provide a more accurate representation of app performance and behavior. This can be done via wifi or USB connection.

The instructions provided cover the essential steps required to clone the Git repository and set up the Routines Android project in Android Studio.

User Guide for Running the Application from User Perspective

Introduction

This user guide will guide you through the process of using the Routines application from updating the contact information to utilize the sms feature, making new people, routines and tasks and then editing them.

Home Screen

The home screen presents with a big R on it.

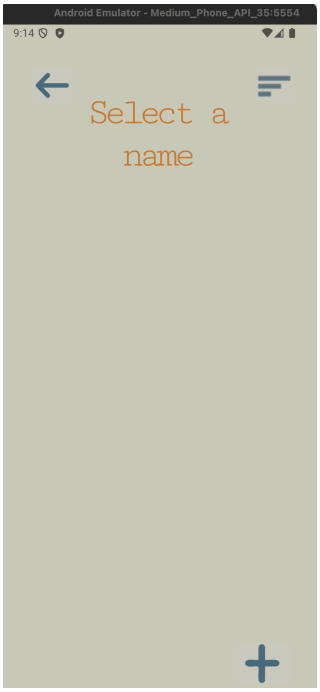
1. Click “Your Routines” to enter into the application

Person List

The person list is shown first with “Select a name” on top

*this page can hold many people’s names

*This page will be empty until the user adds routines.



Contact Page

The contact page is where the user saves a contact phone number for the SMS feature to work. The number will be contacted when a task is check marked completed. Follow this section to first time enter a contact phone number or edit the existing number.

*If the user does not wish for a messaged to be sent, they can leave this page unedited or type in 0000000000

1. Starting from the person list that states “Select a name”
2. Click the menu button on the top right corner
3. A drop down menu will appear
4. Click on “Contact Info”
5. A screen stating “Update phone:” will appear
6. Click in the box showing text or a phone number on the line.
7. Erase the text/number and enter the desired phone number.

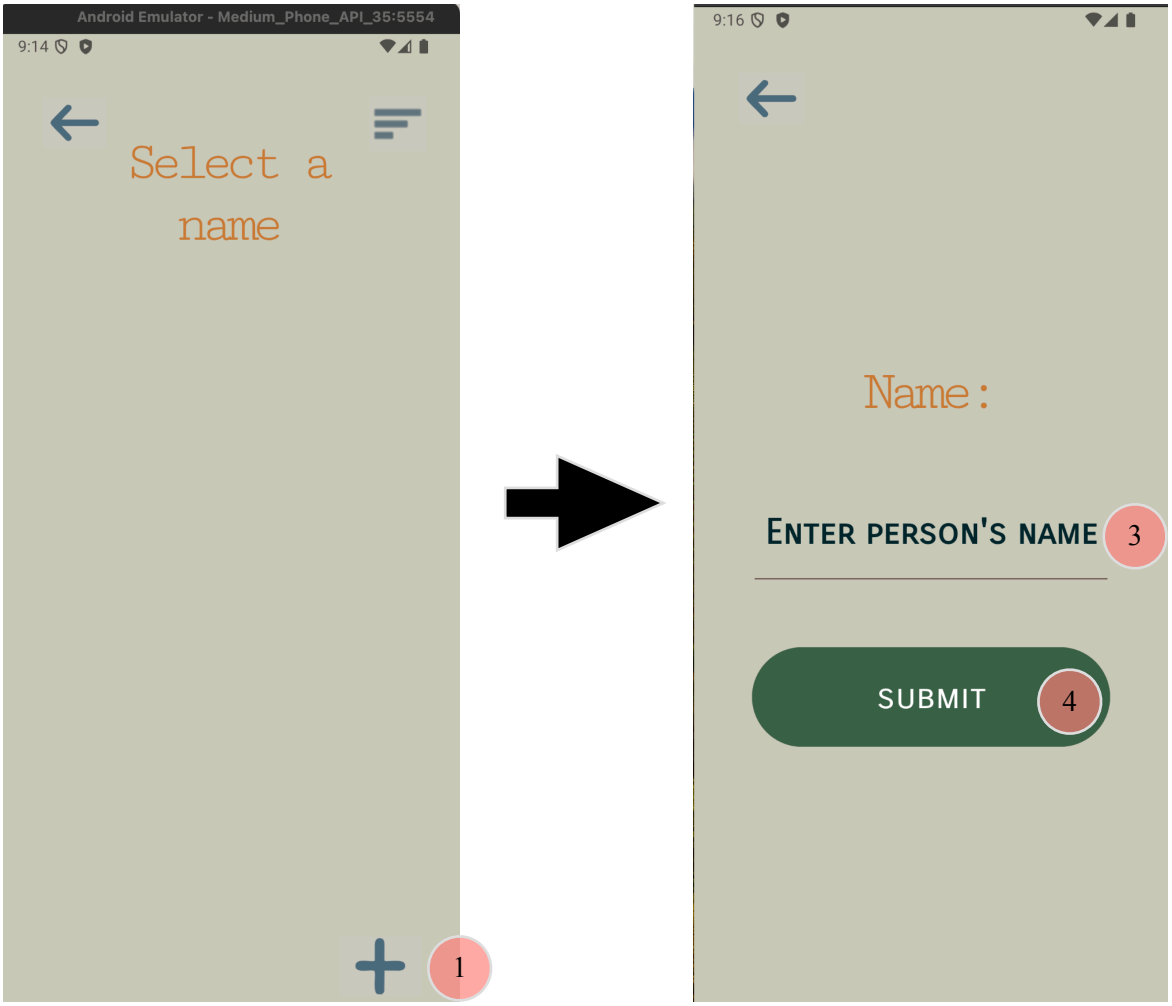
    *This is US based so area code and phone number in 5555555555 form

8. Click the submit button
9. It will return to the select a name screen

New Person

1. Click the plus on the bottom right of the screen
2. This will bring up a new screen that shows Name: and an edit text box that states “Enter person’s name”
3. Type the name in
4. Click the green submit button
5. This will return you to the Select a name screen

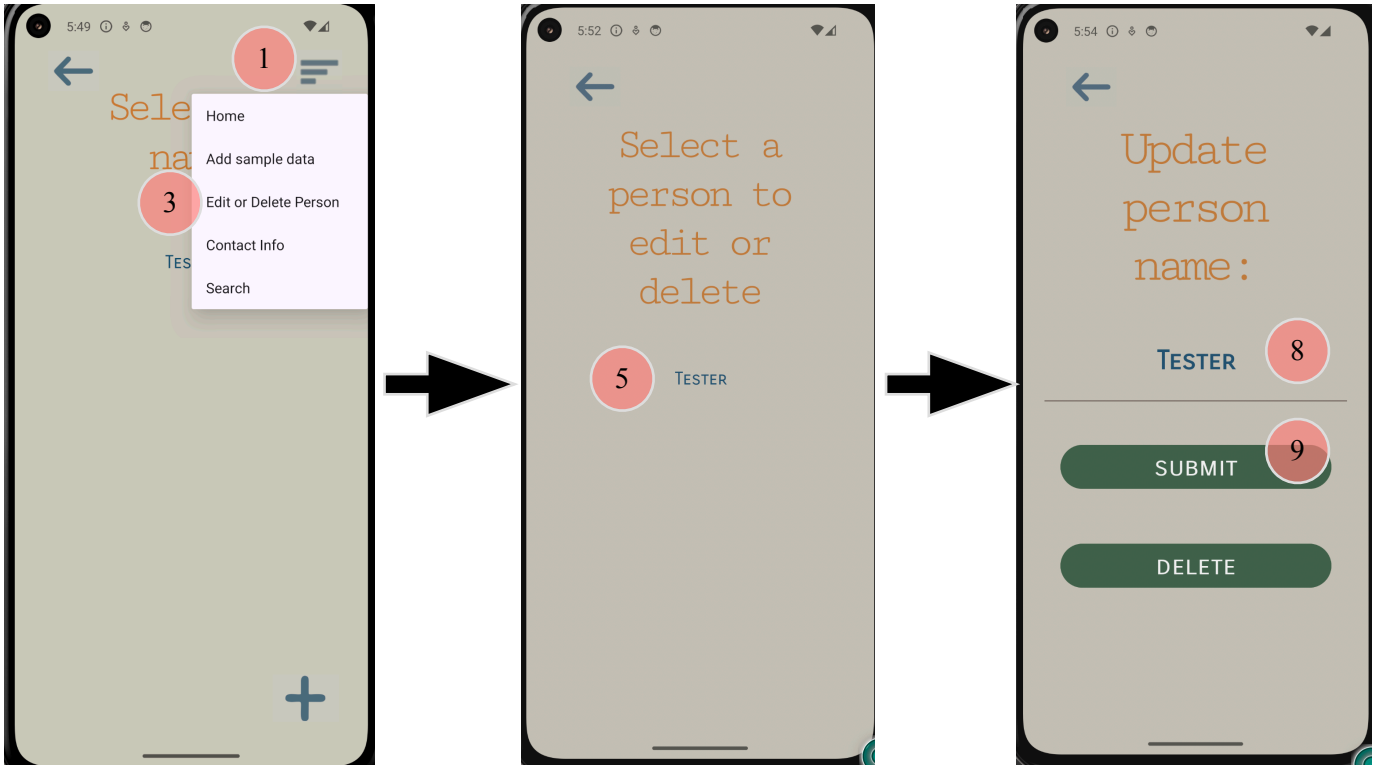
*If needing to exit “the Name: edit screen”, click the top left arrow to back to the person’s list that says “select a name”



Update Person Name

To update a person's name

1. Click the menu lines on the top right corner of the select a name list page
2. This drops down a menu of options
3. Click "Edit or Delete Person"
4. The screen will now show "Select a person to edit or delete"
5. Click the name you want to edit
6. This screen will now show "Update person name"
7. The name clicked will show in the edit text box.
8. Click within the name to edit the text to the updated name
9. Click submit once the name is properly typed in
10. This will return to the Select a name - person list.

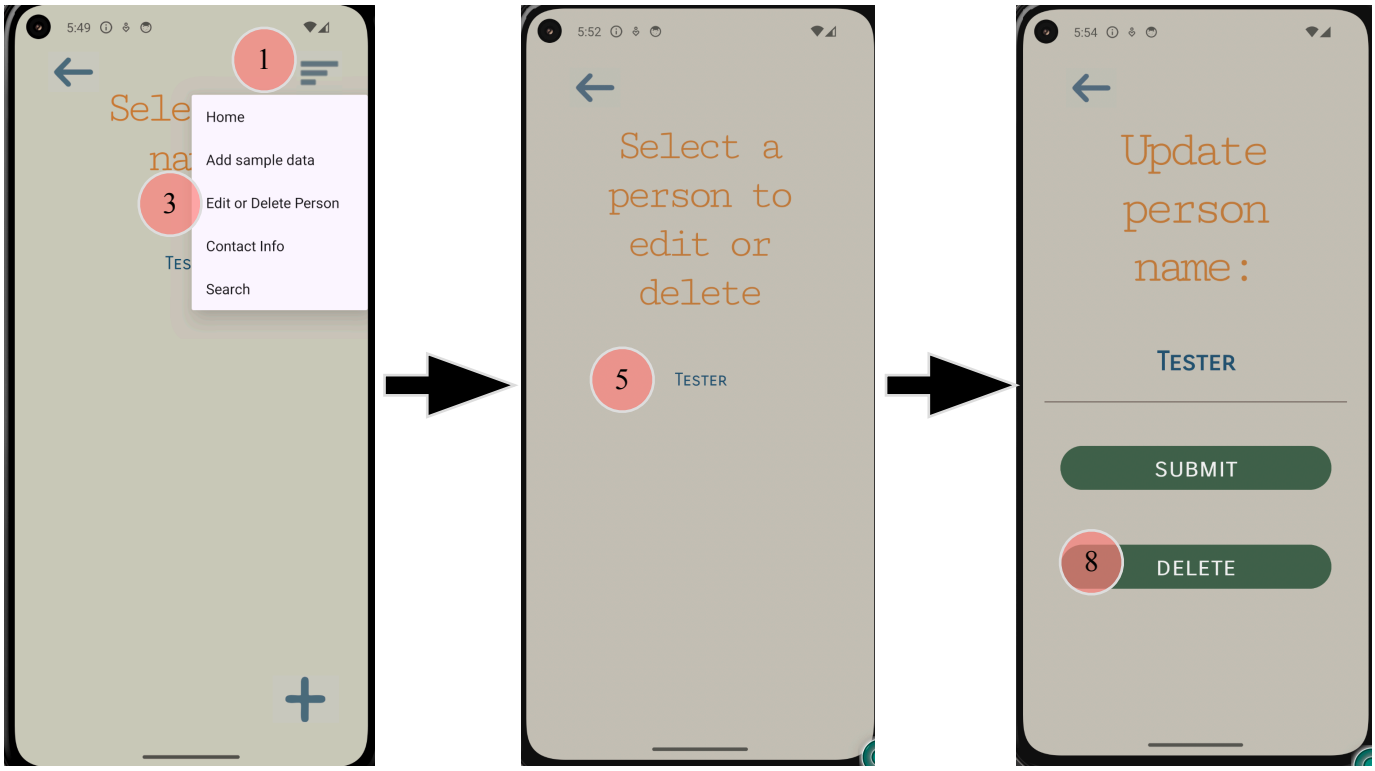


Delete Person

To delete a person's name (This will delete all routines/tasks associated)

1. Click the menu lines on the top right corner of the select a name list page
2. This drops down a menu of options
3. Click "Edit or Delete Person"
4. The screen will now show "Select a person to edit or delete"
5. Click the name you want to delete
6. This screen will now show "Update person name"
7. The name clicked will show in the edit text box.
8. Click the delete button
9. A message stating the name was deleted will appear
10. This will return to the Select a name - person list.

*User can enter as many person names as desired



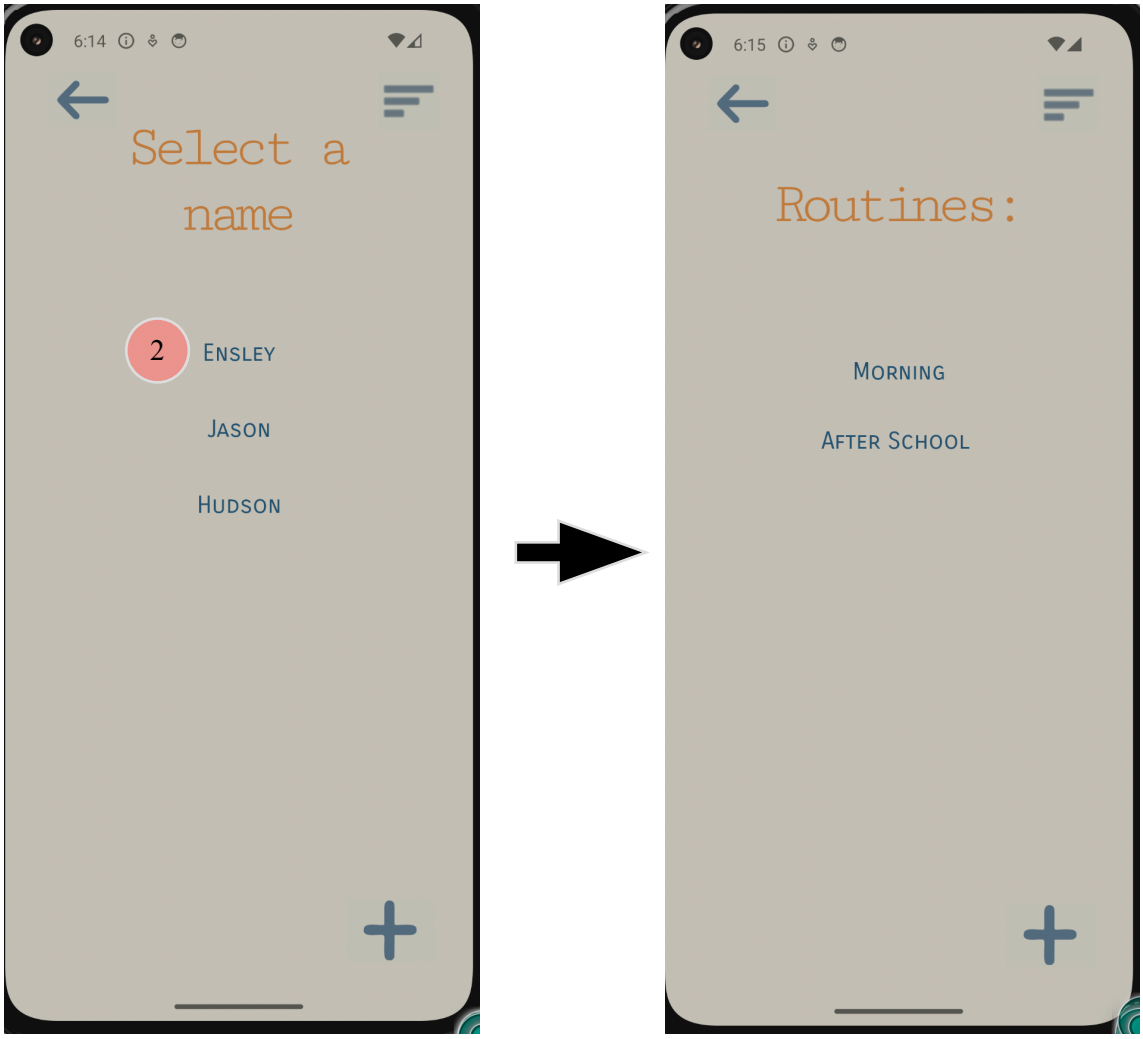
Routine List

To navigate to the routine list

1. Start on the select a name - person list
2. Click the person's name, that routines are desired for
3. Routines: page will appear

*This page will be empty until the user adds routines.

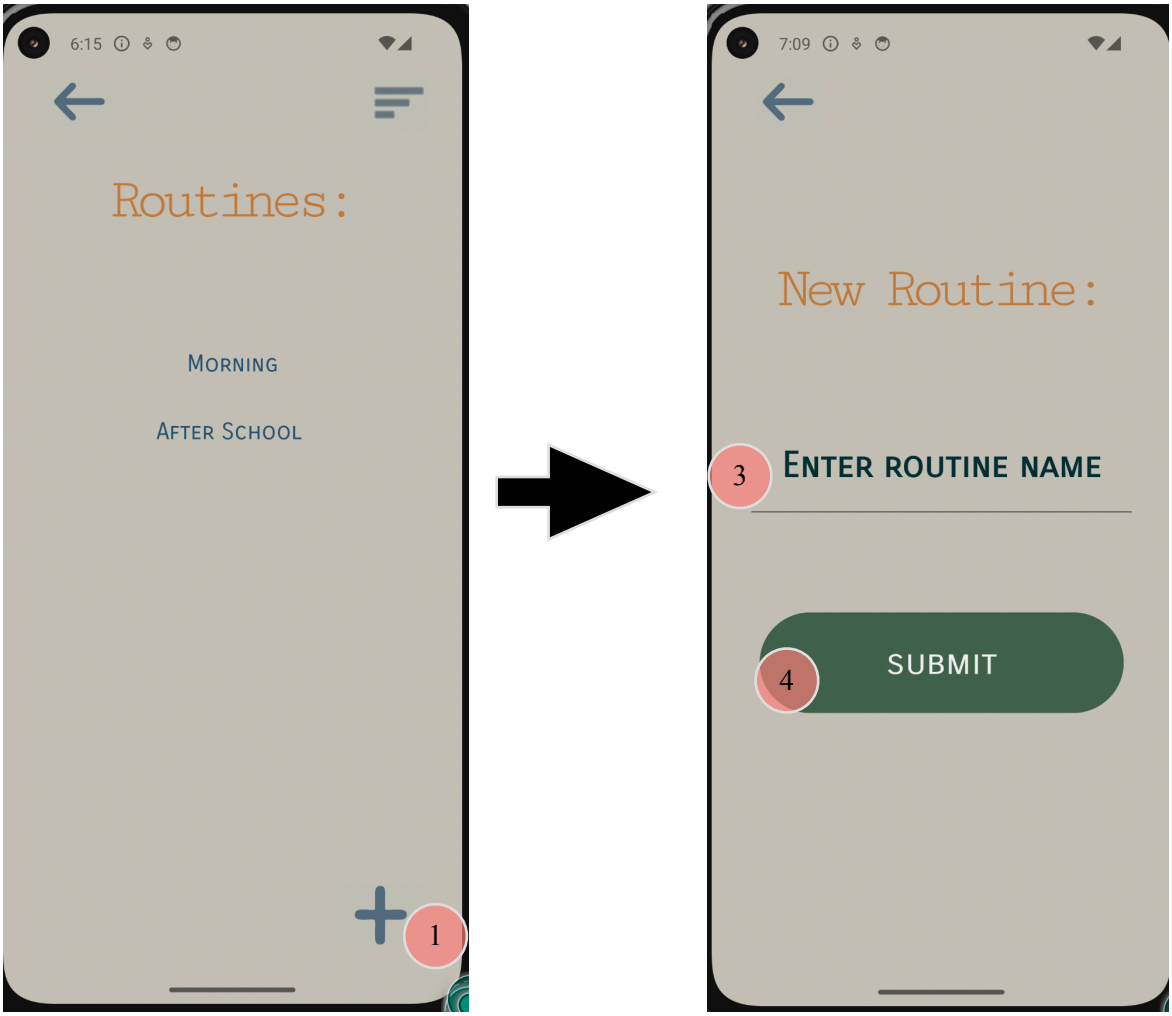
The user can add as many routines as needed



New Routine

1. Click the plus on the bottom right of the screen
2. This will bring up a new screen that shows “New Routine:” and an edit text box that states “Enter routine name”
3. Type the routine name in
4. Click the green submit button
5. This will return you to the Routines screen

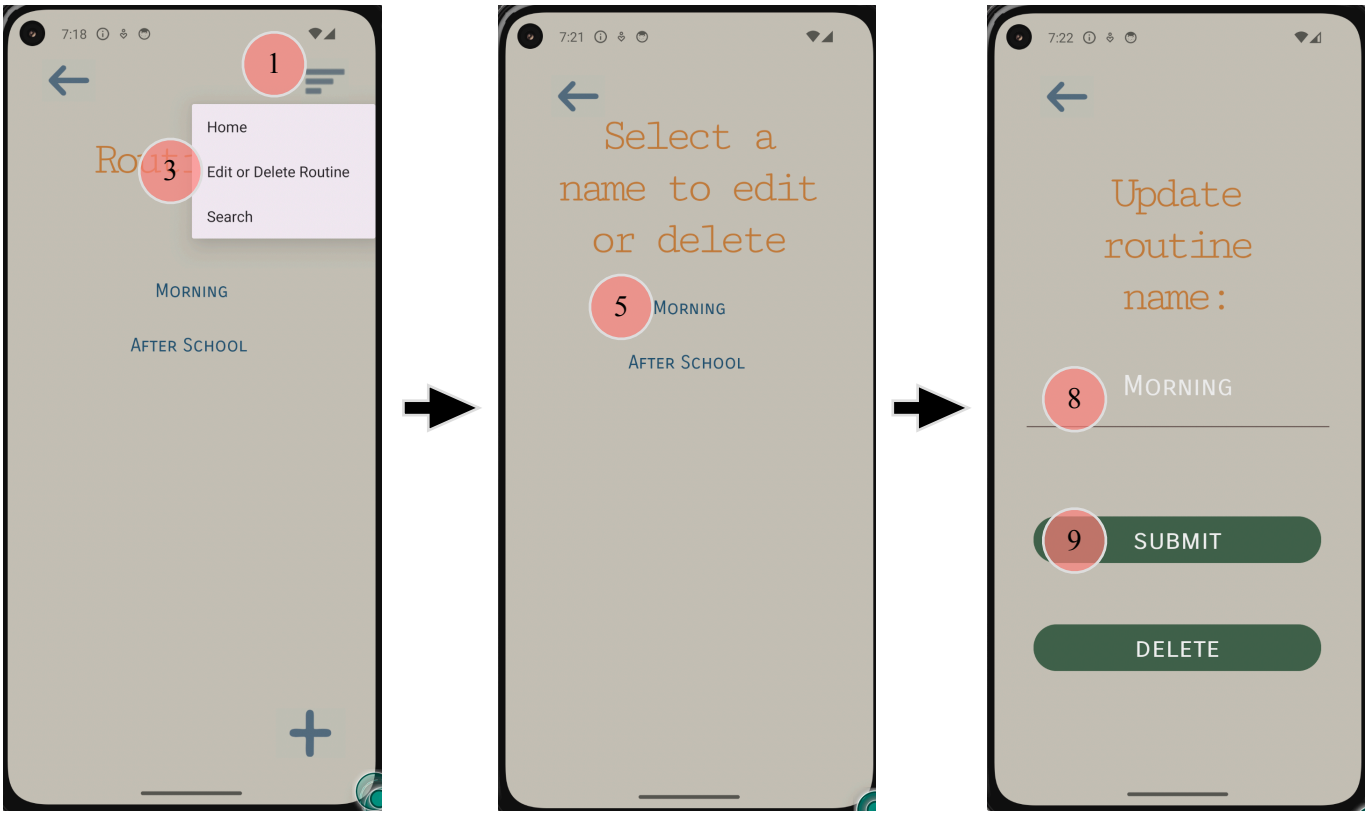
*If needing to exit the “New Routine: edit screen”, click the top left arrow to back to the routine list that says “Routines:”



Update Routine Name

To update a routine's name

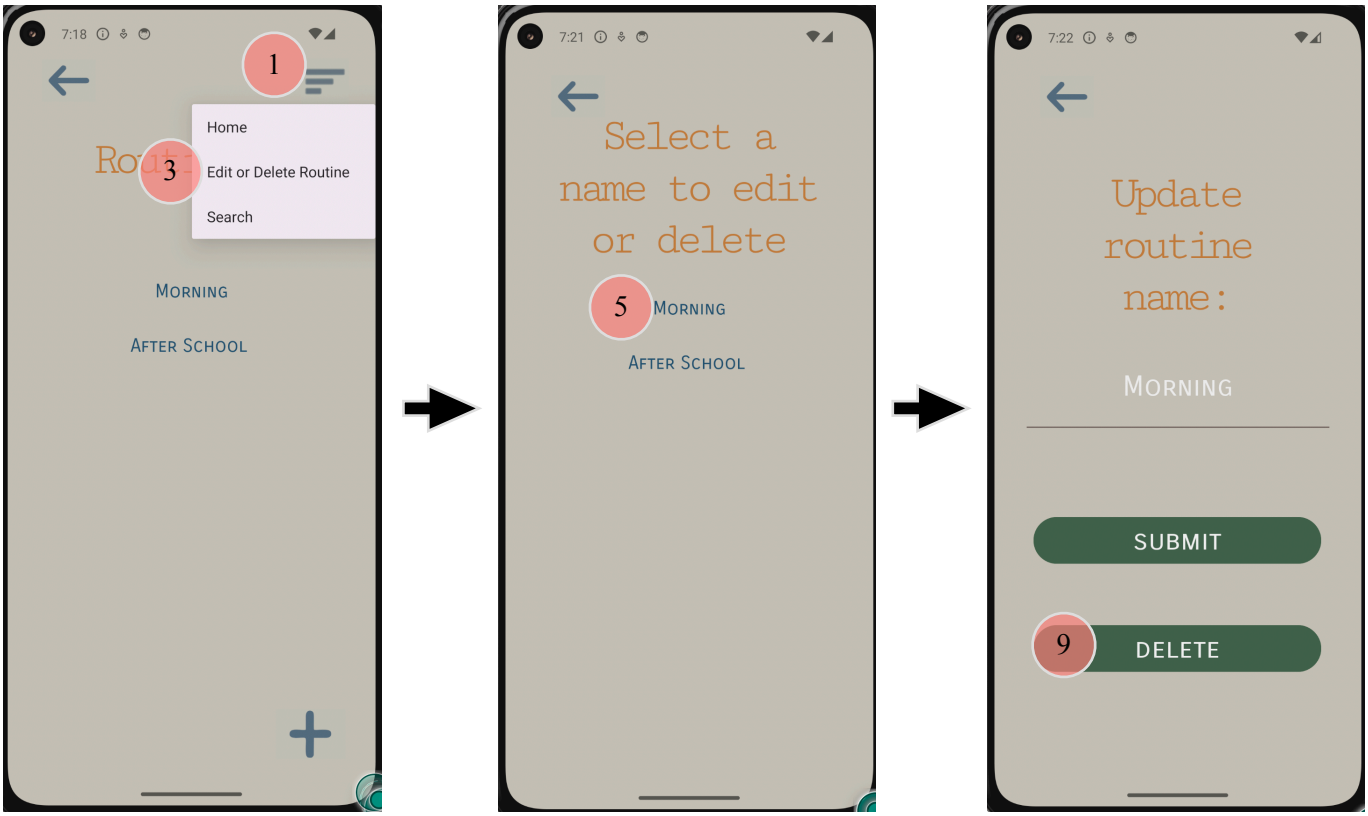
1. Click the menu lines on the top right corner of the Routine: list page
2. This drops down a menu of options
3. Click "Edit or Delete Routine"
4. The screen will now show "Select a routine to edit or delete"
5. Click the routine you want to edit
6. This screen will now show "Update routine name"
7. The routine clicked will show in the edit text box.
8. Click within the routine name to edit the text to the updated routine name
9. Click submit once the name is properly typed in
10. This will return to the Routine: page



Delete Routine

To delete a routine (This will delete all tasks associated)

1. Click the menu lines on the top right corner of the Routine: list page
2. This drops down a menu of options
3. Click “Edit or Delete Routine”
4. The screen will now show “Select a routine to edit or delete”
5. Click the routine you want to delete
6. This screen will now show “Update routine name”
7. The routine clicked will show in the edit text box.
8. Click the delete button
9. A message stating the routine was deleted will appear
10. This will return to the Routine list page



Task List

To navigate to the task list

1. Start on the Routine list page
2. Click the routine name, that tasks are desired for
3. Tasks: page will appear

*This page will be empty until the user adds tasks. The user can add as many tasks as needed

New Task

1. Click the plus on the bottom right of the screen
2. This will bring up a new screen that shows “New task:” and an edit text box that states “task name”
3. Type the task name in
4. Click the green submit button
5. This will return you to the Tasks: screen

*If needing to exit the “New Task: edit screen”, click the top left arrow to back to the routine list that says “Tasks”

Update Task Name

To update a task's name

1. Click the menu lines on the top right corner of the Tasks: list page
2. This drops down a menu of options
3. Click "Edit or Delete Task"
4. The screen will now show "Select a task to edit or delete"
5. Click the task you want to edit
6. This screen will now show "Update task name"
7. The task clicked will show in the edit text box.
8. Click within the task name to edit the text to the updated task name
9. Click submit once the task name is properly typed in
10. This will return to the Tasks page

Delete Task

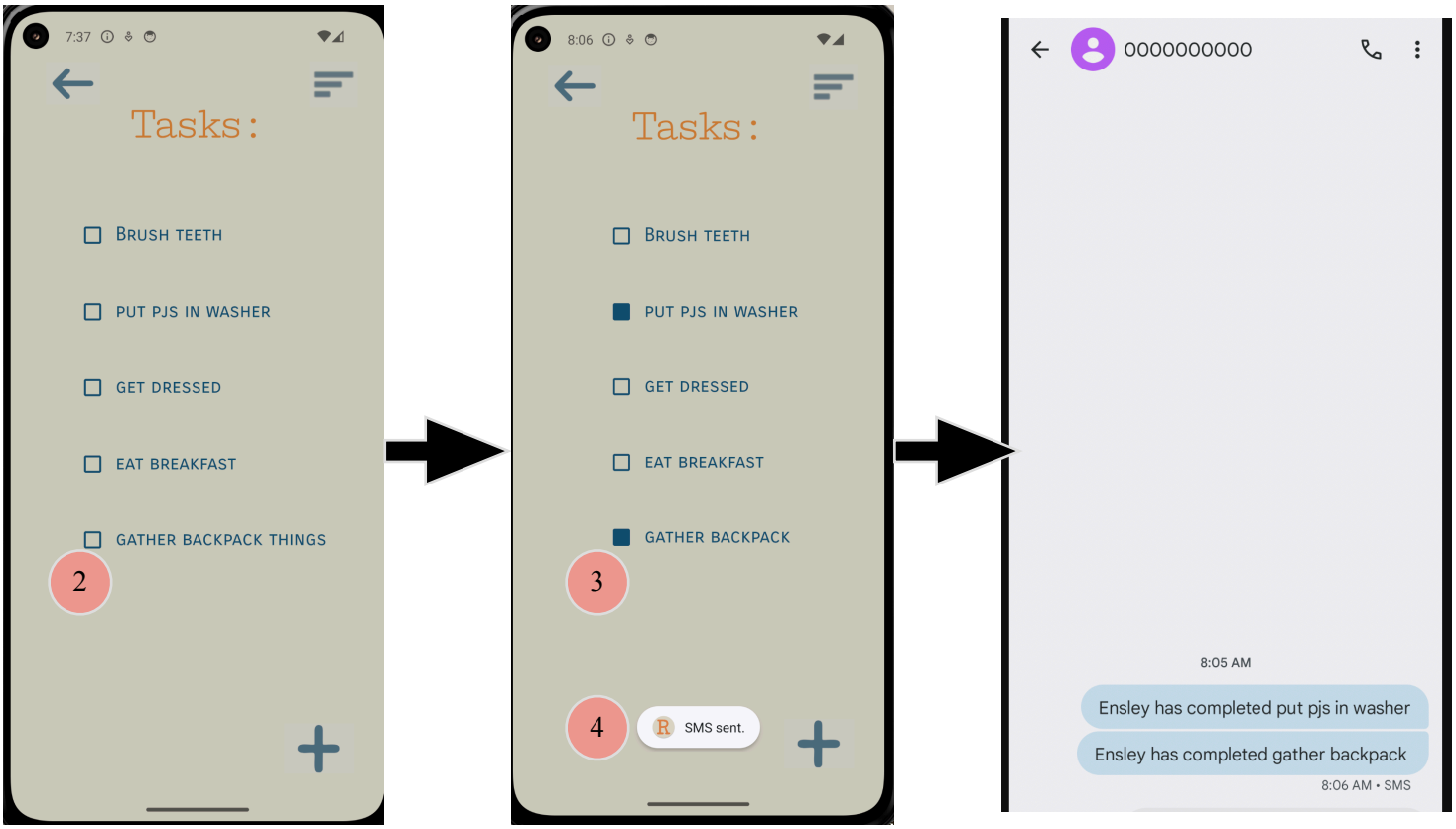
To delete a Task (This will delete all dates associated)

1. Click the menu lines on the top right corner of the Tasks: list page
2. This drops down a menu of options
3. Click “Edit or Delete Task”
4. The screen will now show “Select a task to edit or delete”
5. Click the task you want to delete
6. This screen will now show “Update task name”
7. The task clicked will show in the edit text box.
8. Click the delete button
9. A message stating the task was deleted will appear
10. This will return to the Tasks list page

Check-Marking Tasks

When a task is checked, it marks it as complete for the day and the current date is added to the dates performed page. When the box is checked, it also sends an SMS to the saved contact.

1. Start on the Tasks page
2. Click the checkbox
3. The checkbox will fill in to show it is completed
4. A notification saying SMS sent will appear on the bottom of the screen

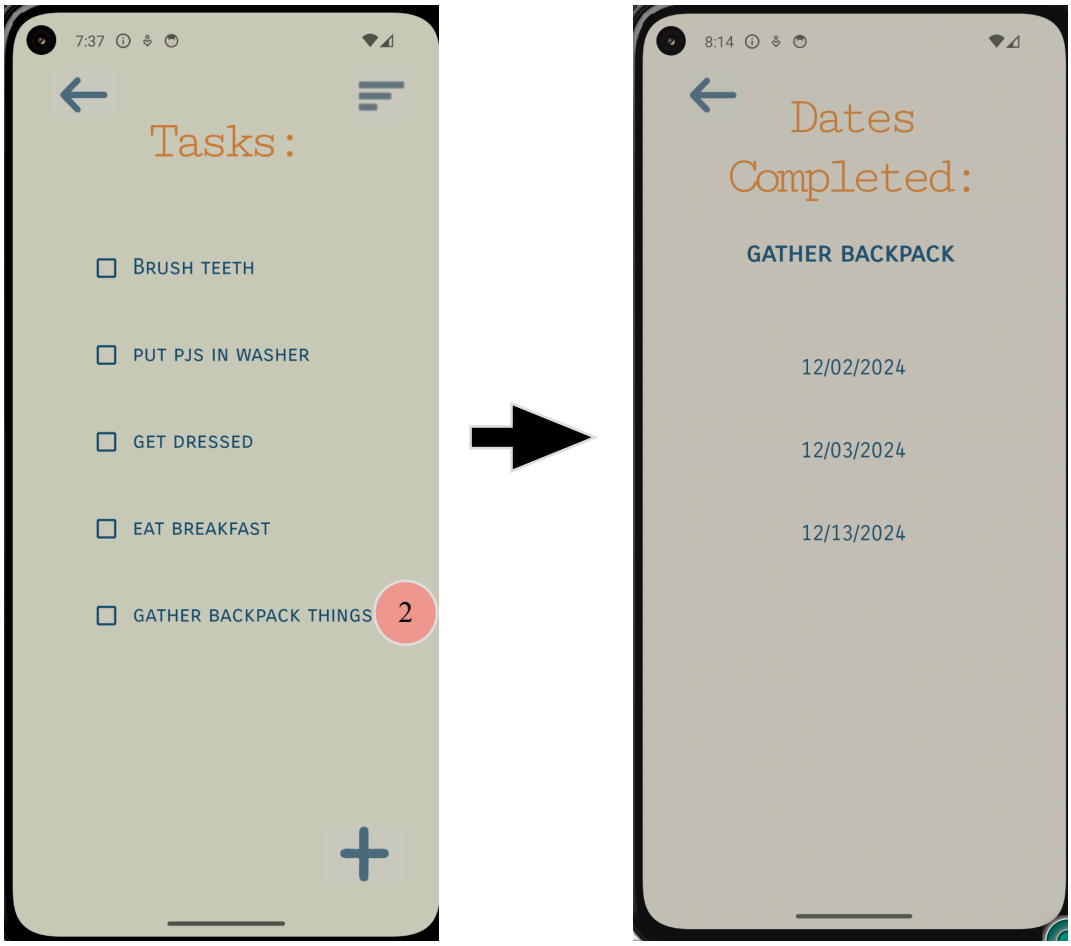


Dates Performed

When the user clicks a task, it marks it here with the date completed

1. Start on the Tasks: page
2. Click the task name the user wants to get the dates for
3. Dates Completed for that task will appear

*This page will be empty until dates have been recorded



Search (Report)

The search feature allows searching for person-task relationship. The letters typed in will bring up a list of people related characters and tasks with related characters.

1. Click the menu button on the top right of the page
2. Click Search option
3. Search page will appear
4. Click the magnifying glass
5. Type in the box the desired query

REFERENCES

No resources to reference