**Simple Scheduler**

Jason Santana, Matthew Pinneo, and Matthew Spitzley

**Introduction**

Simple Scheduler is an Android app for time management. Users can schedule and manage their plans quickly and easily, and never miss a due date with customizable reminders. They can choose to view their schedules as a list of tasks, or on a calendar. Users can choose to synchronize their data by signing in with Google.

**Technology**

1. Android Studio

This is our primary IDE for development. The front end Java code as well as the interface will be developed in Android Studio

1. Android

The OS Simple Scheduler will run on. It is made for Android version 9 or newer.

1. Java

Java is the main language of the front end application.

1. WampServer

For database management.

1. MySQL

For the server side database.

1. PHP

For communicating with the MySQL database.

**Future Work**

In the next version, we want to fully publish our application in the Google API. This will allow non authorized users to have access to the online portion of SimpleScheduler. Also, we would restructure the Task table in the databases to contain an integer id primary key rather than a text field. This would allow for multiple tasks of the same name. We wouldn’t want to change the category table to be this way since we don’t want categories to be able to have the same name.

For the task list screen, we would like to change the add a task functionality from an AlertDialog box to a new activity screen. This would allow for better interface options and management of the task list. The recurring notification option needs to be implemented. The goal is to be able to select any of the options as well as none. The add a task option also needs to be modified to support not selecting a time or date. When changed to a new activity screen, we would like to add better interface options like spinner activities for categories and recurring notification settings.

An option to select an item from the task list and view the fields will be implemented. When viewing a task this way, we will add an option to edit any of the fields for the task and update the task list accordingly.

The adding a category functionality will add categories to the category selection list. A delete category option needs to be implemented so we can update the category options.

The history screen still needs to be implemented. Similar to the task list, a history list needs to be created with tasks marked complete. Along with this goes the deleteTask function in TaskList.java. When a task is marked complete, that task needs to be added to the history list as a task with completion boolean set as “true”. An option to restore tasks from the history list was also an initial goal but not able to be implemented in time.

We would like general interface upgrades. Things like better menu onClick functions, combining the date and time selecters, and personalizing the app more.

Adding a Yearly View to the Calendar screen was meant to be a part of the final build, but due to more pressing issues this feature had to be pushed back to the future work.

A functionality needs to be implemented to display tasks with no date. Initially we wanted the calendar view to show highlighted days that had tasks scheduled but were unable to implement in time. Along with this, we want to be able to select a day and display the tasks set to that date. This is partially implemented but needs to be improved upon.

Adding push and email notifications was going to be a feature of the first release, but due to time constraints we decided not to implement them.

Setting tasks to recur was also going to be a main feature, but we decided to cut it as it caused issues to arise late in testing.

**Known Bugs**

Our project is registered as a test project in the Google API. This means that only users added to the test group can be signed in. This effectively makes the online portion of the app undeployable. Also, we ran into issues getting a web server, and had problems connecting our app to a local database. As a result, we have been unable to run our php scripts.

The database structure has an issue that lowers its integrity. Since the primary key of the Task table is a text field, if multiple tasks had the same name, issues would arise. This should be fine for the Category table as you can not have two of the same categories.

Currently there is an issue with selecting the recurring notification option. The recurring notification field would work similarly to the category selection however, when nothing is selected there is an error with setting recurring notifications to none or null.

When setting a date there is not a check for selecting a date in the past. There will be an input check when selecting the date so that it does not occur before the current date.

There is a bug with the tasks marked as complete. Initially we were adding the task back with the boolean value changed and then trying to display based on that value. The task continued to appear in the task list regardless of completion status. To resolve this we need to add the completed tasks to a separate list of tasks and display the history list and task list individually.

**Deployment**

Since the project is a test project in the Google API, the database isn’t really accessible from the application. However, to deploy the app on our machines we created the database with the following steps. We are using WAMP server and phpMyAdmin to manage the database. The database user should be named “root” with no password and have full permissions. There are two create table statements and one constraint statement:

CREATE TABLE `test-example`.`category` ( `category` VARCHAR(255) NULL DEFAULT NULL , UNIQUE (`category`(255))) ENGINE = InnoDB;

CREATE TABLE `test-example`.`task` ( `task\_name` VARCHAR(255) NOT NULL , `category` VARCHAR(255) NULL DEFAULT NULL , `time` TEXT NULL DEFAULT NULL , `recur` TEXT NULL DEFAULT NULL , `email\_notification` TEXT NULL DEFAULT NULL , `push\_notification` TEXT NULL DEFAULT NULL , `completed` INT NOT NULL , PRIMARY KEY (`task\_name`(255))) ENGINE = InnoDB;

ALTER TABLE `task` ADD CONSTRAINT `category\_constraint` FOREIGN KEY (`category`) REFERENCES `category`(`category`) ON DELETE SET NULL ON UPDATE SET NULL;

Then the PHP scripts located in SimpleScheduler/SimpleSchedulerApp/php\_scripts should be placed in the www directory in the WAMP server.

**Classes and Scripts**

The functions in the Task class will be as follows: a constructor, getName, setName, getCategory, setCategory, getTime, setTime, getRecur, setRecur, getEmail, setEmail, getPush, setPush, isComplete, setComplete. The constructor takes each parameter (String name, Category category, ZonedDateTime time, Recur recur, ZonedDateTime email, ZonedDateTime push, and boolean complete) and assigns them to the private fields in the class. The getName function will return the task name as a String. The setName function will take a String input and replace the Task’s name. The getCategory function will return the Category object assigned to this Task. The setCategory function will take a Category object to set the Task’s category. The getTime function will return a ZonedDateTime which is the due date of the Task. The setTime function will take a ZonedDateTime to set the Task’s time. The getRecur function will return a Recur enum object. The setRecur function will take a Recur enum object and set when or if the task will recur. The getEmail function will return a ZonedDateTime that is the time when the user will receive an email notification for the given Task. The setEmail function will take a ZonedDateTime and set when the email notification will occur. The getPush and setPush functions will act the same as the email functions, but will return and take ZonedDateTime fields which gives the task a time for a push notification. The isComplete function will return a boolean that tells if the task has been completed. The setComplete will take a boolean and change the completion status of the task. The completion status of the task will be used to separate the tasks to be displayed on the list and calendar screen and the tasks to be displayed in history.

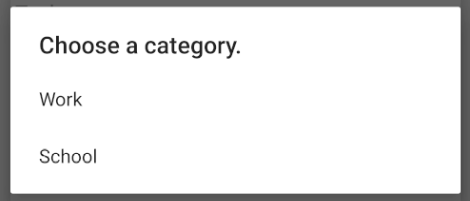
Also within TaskClass.java is the Recur enum. It is an enum of Strings corresponding to the times when the user can set a task to automatically recur. The options are: Daily, Weekdays (Monday through Friday), Weekly, Monthly, and Yearly.

The Category class has only one String field “name”, a constructor, and functions getName and setName. The constructor has a name parameter. The function getName returns the category’s name, and setName changes the category’s name.

MainActivity class, also known as the Task List, is the default activity that launches in the app until the user changes it to the Calendar screen in the settings. The onTouchEvent method in the MainActivity takes the user to the CalendarScreen class if the user swipes right and to the History class if left. This is achieved by using variables xBegin and xEnd, which record the beginning and ending x coordinate of the touch event, then compares the values in order to determine if the user has swiped right or left.

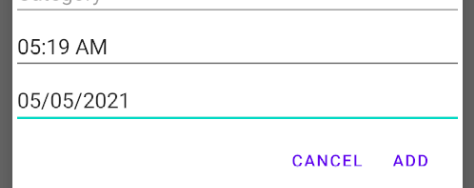
In the main class TaskList we have the methods onCreateOptionsMenu, onOptionsItemSelected, updateUI, AskOption, and deleteTask. The onCreateOptionsMenu inflates our “Add Task” menu and calls onCreateOptionsMenu. The onOptionsCreateOptionsMenu function uses an AlertDialog pop-up to display a linear layout of EditText fields. There are EditText fields for name, category, time, date, and recurring notifications. When inputting the name, the text is then set to a variable that is later used to create the task.. The category EditText field uses another AlertDialog to bring up a list that is populated with the getCategoryList DBHelper method. Whenever a category is created, the category will appear on this option list. To confirm the category, select the positive button. Selecting this button will assign the task category to the option selected.When selecting a time, clicking on the time EditText field brings up a TimePickerDialog window with a clock to select the time for completion of a task. When selecting a date, clicking on the date EditText field brings up a DatePickerDialog window with a calendar to select a date for completion of a task. These two fields are then taken and converted to a string which is then parsed to format it to the correct ZonedDateTime to pass into the task object. Currently, the recurring notification field is missing. In the future, we would like to have functionality for when selecting the recurrence EditText field an AlertDialog box appears with the options for selecting when notifications should occur. After the user inputs the task fields, selecting the add button takes all the fields and creates a task. This task is then added to the task list.

When the category EditText onClickListener is used, an Array list is created with the getCategoryList DBHelper function. A string array is created to the size of the created ArrayList. A for loop is then run to assign string values of Category names to the string array. These values are then used to populate the category options list. Each time a category is added the list will reflect the added category. When an option is selected, the text for the category EditText field is set to whichever category it is from the list. The category select is then added to the linear layout. The window below is an example of the choose category box. The recur window operates the same way.



When the time EditText field is selected, the onClickListener brings up the TimePickerDialog. The TimePickerDialog brings up a clock that allows the user to select a time that they wish for the task to be completed by. The clock displays based on the settings preference for 24 or 12 hour time. If it is set to 12 hour time an AM/PM option is added to the clock. In order to make the selected time compatible with the ZonedDateTime type required for adding a task, if statements are used to convert single digit hour and minutes to display as double digit. For example, when “7:30” is entered, it is converted to “07:30”. The text field is then set to the selected time and the entire field is added to the linear layout.

The date field is handled in a similar way as the time field. An onClickListener has been applied to the date EditText field. This brings up the DatePickerDialog, which displays a calendar for the user to select a date they wish the task to be completed by. Again, because of the ZonedDateTime format assigned to tasks, if statements are used to convert single digit months or days to display with a “0” as the first position. The date is then set as the text for the EditText date field. The time and date are entered separately and converted to a single date time string that is parsed and used to create a ZonedDateTime variable used to create a task. The image below displays the format that single digit times and dates are displayed as.



The add task AlertDialog box has a set positive button and set negative button. Negative cancels and dismisses the add task window. When selecting “Add”, which is the title of the set positive button, all of the values entered into the EditText fields are retrieved and used to create a TaskClass object “mTask”. The task name is taken from the first task EditText, where the string input by the user is set to the task name. The time and date string options are concatenated into a format that the ZonedDateTime parse function is compatible with. This process uses DateTimeFormatter to designate the format of the ZonedDateTime. The ZonedDateTime item is then assigned to the tasks completion time. Currently the option for email and push notifications will not appear if notifications are turned off in settings. When all the fields are complete and the task has been added the updateUI function is called to update the list.

Marking a task complete uses a button that calls the deleteTask function in the TaskList class. This function uses an AlertDialog window to ask the user for confirmation of task completion. The deleteTask function calls the AskOption function which brings up a window with a set positive button “Complete” and negative button “Cancel”. When the user selects “Complete”, a call is made to the database to remove the task from the task list. The updateUI function is called so the Task List screen reflects the removal of the task.

The updateUI function is used to get the task list and display any changes to the task list on the task list screen. After a task has been added this function is called to make a query to the database to retrieve the new task list. It then uses an adapter to display the updated list to the task list screen.

The CalendarScreen class is a repurposed TaskList that displays the same information in a calendar format. In the onCreate method the calendar defines calendar, dateView, noDate, settingsBtn, and signInBtn variables in accordance to what is in the layout file. It then gets the shared preferences so the date at the top of the screen can display correctly according to how the user would like to see the date formatted. There is an onSelectedDateChangeListener that displays all of the tasks on that day, as well as changing the date at the top to reflect that day. Continuing in the onCreate method is an onClickListener for the noDate button that gets the list of all tasks that have no existing date, and also changes the date at the top to instead say “No Date.” There are two more onClickListeners for the settings button and the google sign in button, which each just open their respective activities.

Outside of the onCreate method is the onResume method which is meant to update the UI in relation to any user made changes in the settings. This is here because without it, if a user were to change the date format then went back to the calendar screen, the date at the top and in the task list would still reflect the previous date format until a new day was selected or a new task was added. The last method in the CalendarScreen class is the onTouch event that will lead to the History screen if the user swipes the screen right and will take the user to the Task List screen if the user swipes left.

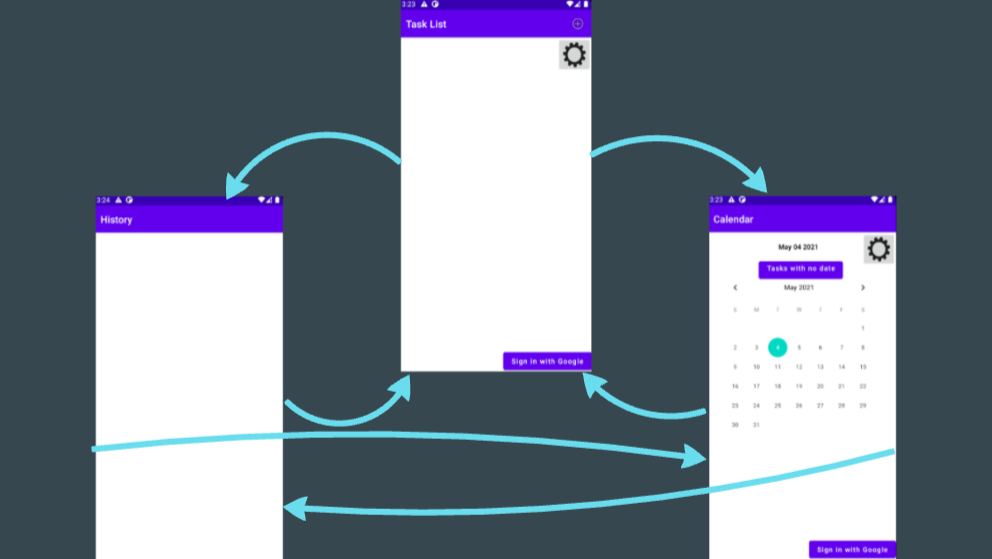
GoogleSignIn class is accessed by the CalendarScreen or MainActivity class. Using all of Google's sign in classes allows users to sign into their Google account. Starting at the onCreate method a GoogleSignInClient and a GoogleSignInButton is created. The GoogleSignInButton onClick method creates the signInIntent where the user enters their information to sign into Google. The handleSignInResult method is where it decides what to do next depending on how the sign in attempt went. Upon successful login the user will be brought to the ProfileActivity class, and if it fails will give a user response telling them they failed to log in.

The ProfileActivity class displays to a user their profile image, name, email, and a sign out button. The image, name, and email are not interactable, but the sign out button is. In the same way the sign out button worked, there is an onClickListener that attempts to sign out the user, and when it is successful will bring the user back to the previous GoogleSignIn class.

The SettingsActivity class exists purely to show the root\_preferences.xml. One of the first preferences is the “help” preference, which on click creates an alertDialogBuilder which gives users a UI to select what specific topic they would like help with. Depending on what is chosen a switch statement gives the user a new alertDialog with text that is either a helpful description or a statement of how to navigate the topic that was chosen. For example when the user selects “Task” there is dialog describing what tasks are in our program, and if the user selects “Task List Screen” the dialog tells the user how to navigate through and use the “Task List Screen.” There is a dialog for each of the following topics: Tasks, Categories, Task List Screen, Category Screen, Calendar Screen, History Screen, Settings Screen.

The second preference in the root\_settings is the starting screen. This is a switch preference, when it is false the task list will be the screen that the user first sees, and when true the calendar will be the first screen the user sees at app startup. In order to make this work, there is a LauncherFakeActivity class that is set as the MAIN and LAUNCHER in the AndroidManifest.xml. When this activity starts it will immediately either redirect to the TaskList activity or the CalendarScreen activity depending on if that preference is set to false or true respectively.

The third preference is another switch preference which controls how time is displayed to the user, either 12 (false) or 24 (true) hour time. For this to work correctly the preference has to be passed to the method toggleTime(boolean) found in the Settings class. It is important to note the Settings class is different from the SettingsActivity class. The toggleTime method returns a string to be used for formatting throughout the methods in the app that display time to the user. The fourth preference, Date Display Format, works the exact same way with one difference, which is that it is a list preference instead of a switch preference. Because of this the method dateDisplay(int) takes the preferences selected index as an input, where there are 6 different choices to be made.



The very first thing a brand new user will see is the Task List Screen. On this screen, at the top right is an add task button and a settings button directly underneath it. The entire middle section of the screen is the list of all existing tasks that are not yet completed. At the bottom right is a button called “Sign in to Google” where the user is brought to the GoogleSignIn activity. After a user signs into google the GoogleSignIn activity will redirect to the Profile activity until the user signs out again. From the Task List screen, to move to the Calendar Screen the user needs to swipe across the screen right, and to get to the History Screen will need to swipe left.

The class DBHelper extends SQLiteOpenHelper, and is responsible for all SQLite operations. It has fields for each of the table and column names: *CATEGORY\_TABLE, COLUMN\_CATEGORY\_NAME, TASK\_TABLE, COLUMN\_TASK\_NAME, COLUMN\_TASK\_CATEGORY, COLUMN\_TASK\_TIME, COLUMN\_TASK\_RECUR, COLUMN\_TASK\_EMAIL\_NOTIFICATION, COLUMN\_TASK\_PUSH\_NOTIFICATION,* and *COLUMN\_TASK\_COMPLETE.* It also has fields for the database name and the version: DB\_NAME and DB\_VERSION.The methods are: a constructor, onCreate, onUpgrade, addCategory, addTask, deleteCategory, deleteTask, fetchCategoriesParse, fetchTasksParse, syncCategoriesParse, syncCategoriesParseHelper, syncTasksParse, syncTasksParseHelper, getCategoryList, and getTaskList, which has 2 overloaded methods. The constructor takes a Context object, which is information about the current app environment. It names the database and version.

The onCreate method is called when the database is first created. It takes a SQLiteDatabase instance. The SQL table creation statements are written and executed in this method. The onUpgrade method takes a SQLiteDatabase instance, and the old and new version numbers as integers. It drops the current tables and then calls the onCreate method. The addCategory method takes a Category object and adds it to the database. It returns a boolean value true if the table insertion was successful, and false if it was not successful. The addTask method takes a Task object and adds it to the database. It returns a boolean value true if the table insertion was successful, and false if it was not successful. The deleteCategory method takes a Category object to be deleted. It deletes a Category from the database using its name. The deleteTask method takes a Task object to be deleted. It deletes a Task from the database using its name.

The fetchCategoriesParse method takes a Context and a RequestQueue. A RequestQueue is from the Android Volley library and is a queue of network requests. The method calls the php script “fetchCategories.php” which gets a json array from the MySQL database. It then parses an entry of the json array into a Category object and adds that Category to the database, for each category in the json array.

The fetchTasksParse method takes a Context and a RequestQueue. The method calls the php script “fetchTasks.php” which gets a json array from the MySQL database. It then parses an entry of the json array into a Task object and adds that Task to the database, for each category in the json array.

The syncCategoriesParse method takes a RequestQueue. It creates a String in json format that contains the entire Category List from the SQLite database. It then calls the syncCategoriesHelper method to continue the operation. The syncCategoriesHelper method calls the php script “syncCategories.php” which uses an INSERT SQL statement to insert the json object into the MySQL database.

The syncTasksParse method takes a RequestQueue. It creates a String in json format that contains the entire Task List from the SQLite database. It then calls the syncTasksHelper method to continue the operation. The syncTasksHelper method calls the php script “syncTasks.php” which uses an INSERT SQL statement to insert the json object into the MySQL database.

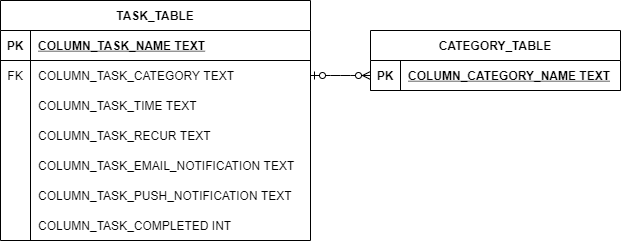
The getCategoryList function returns an ArrayList of Categories from the database. The getTaskList function returns an ArrayList of Tasks from the database. The data from the database columns is parsed into the appropriate types to create Task objects to be added to the ArrayList. The two overloaded getTaskList functions show TaskLists based on certain parameters. The first takes a ZonedDateTime to show a task list for a given day. This is used in the calendar screen. The next getTaskList function takes a String and shows tasks with no date assigned to them. This is also used in the calendar screen.

All 4 PHP scripts are located in SimpleScheduler/SimpleSchedulerApp/php\_scripts. They are: fetchCategories.php, fetchTasks.php, syncCategories.php, and syncTasks.php. The script fetchCategories makes a connection to the database, fetches the table rows from the MySQL database category table, creates an array, and encodes it as a json to be echoed. The script fetchTasks makes a connection to the database, fetches the table rows from the MySQL database task table, creates an array, and encodes it as a json to be echoed. The syncCategories script makes a connection to the database, defines the structure of the POST json array, then runs a query to insert it into the MySQL category table. The syncTasks script makes a connection to the database, defines the structure of the POST json array, then runs a query to insert it into the MySQL task table.

**Diagrams**

Class Diagram

<https://drive.google.com/file/d/1E_klkFojsVQt-JwTtb2h5M_CnGv4n9Om/view?usp=sharing>

SQLite

MySQL