LAU
الجامعة اللبنانية الأميركية
Lebanese American University

SPRING 2022

Software Testing and Implementation Document

for

LAU Virtual Learning Assistant

Based on LAU VLA SRS Version 1.0

Report by Osama Shamout and Omar Mlaeb

May 18, 2023

Document Reference Number: OPENG/D&A-LAU-VLA-V1.0

Copyright Notice

Founded in 2022, Op-Engineers Ltd., had its humble beginnings in the heart of Beirut at the Lebanese American University campus. The company has been an extraordinary technological service company providing innovative technological solutions in the fields of Software Engineering and Software Development. By following the ISO/IEC/ STANDARD IEEE 29148, Op-Engineers Ltd. was able to establish its presence by providing high-quality products that fulfill the market needs.

Proudly Engineered in Lebanon.

This page intentionally left blank.

Table of Contents

1. Introduction

The purpose of this Testing and Implementation document is to showcase our VLA system in execution. First, since most LAU students prefer using their mobile phones and have it more accessible to them. We saw the need to implement the application on the Android platform with plans to expand over to iOS to which shall assume about 98% of the Operating Systems of smartphone in the year of writing this document.

The IDE used in developing this application has the following specifications:

Android Studio Bumblebee | 2021.1.1 Patch 2
Build #AI-211.7628.21.2111.8193401, built on February 17, 2022
Runtime version: 11.0.11+0-b60-7772763 aarch64
VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.
macOS 12.2.1
GC: G1 Young Generation, G1 Old Generation
Memory: 2048M
Cores: 8
Registry: external.system.auto.import.disabled=true
Non-Bundled Plugins: org.intellij.plugins.markdown (211.7142.37)

Second, due to time constraints, we used Firebase tool from Google Developer to establish database connection and simulate the real dataset. Firebase integrates with Android Studio seamlessly and provides great features that reduced the labor in setting up the PHP (APIs) for the system significantly. In Firebase, the user data included email addresses in the LAU email format username@lau.edu for students, and username@lau.edu.lb for faculty. This will be of important usage later in the system.

Third, in our VLA system, we simulated the cases for an average correct user input to validate that the requirements of the VLA system are met. Also, we tested for abnormal input to detect anomalies using defect-testing. Various invalid input was fed to the system in order to decrease any possibility of error when the system is released. Hence, the idea implemented was to validate information extensively in every step before submitting it to the database in order to eliminate errors stemming from invalid data retrieval (to be discussed later in Section 2). Finally, after inspecting the and analyzing the code of the application, we verified some of the requirements for

the VLA system from the implemented Use-Cases. Hence, completing the final stage of the product's development (V&A).

Finally, the system is expected to function fully with the following functionalities, Log-In, Create Task, Task Track. Meanwhile, the system has been partially developed to encompass Registration which is restricted to Administrators (more details in the System Implementation).

2. System Implementation

2.1 Log In

The Log In in the VLA system is based on the Log In Use-Case previously mentioned in the document "D&A LAU VLA SRS V1.0". This Log In activity can be accessed from the main page (First Activity of the Application) by clicking "Get Started!" (See Figure 1.0). After, pressing on "Getting Started!", the system moves to the Log In page. (See Figure 2).

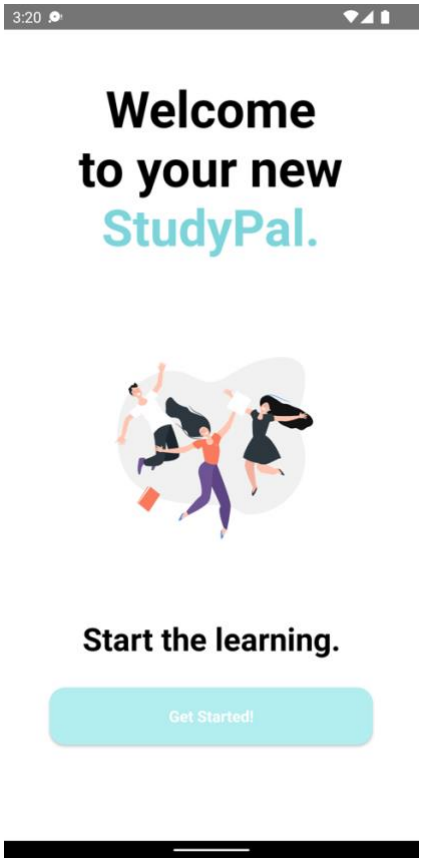


Figure 1: Main Page of the VLA System.

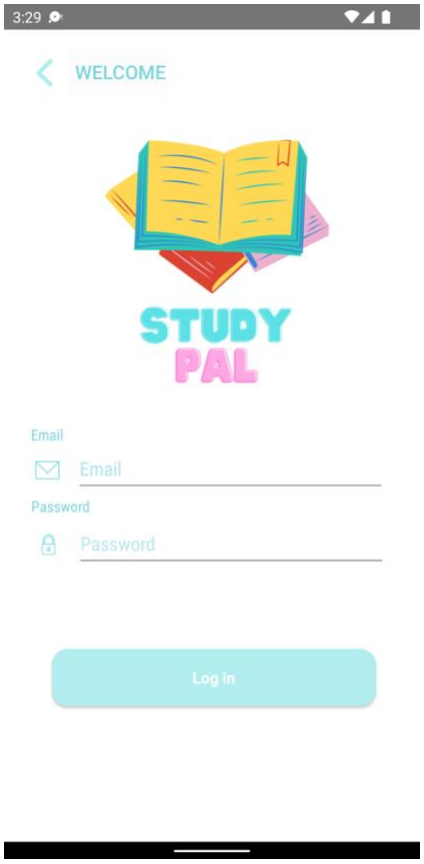


Figure 2: Log In Page of the VLA System

After invoking the Log In Java class from the main screen by the following intent

```
public void OnClickGetStarted(View view){
    Intent intent = new Intent(this,LogIn.class);
    startActivity(intent);
}
```

The Log In page will have the variables and methods below:

Variable	Use
private FirebaseAuth mAuth;	To communicate with the Firebase Database to establish user authentication.
boolean isInstructor = false;	A flag to know if the user is an instructor or not.
EditText input_email; EditText input_password; TextView dialogue; Button log_in;	Initalize the Views created in the Android Studio Log In page.

Methods	Use
public void onBackPressed()	Override on back. Restricts user from pressing back and going to the log-in area, or vice versa.
public void OnClickReturnToWelcome	Enables user to go back to the welcome (main) screen.
public boolean checkInstructorStatus(String email)	Checks if the e-mail input is an instructor's e-mail or not.
public boolean checkPassword(String pass)	Checks if password input box (EditText) is empty.
public boolean validateEmail(String email)	Checks if the e-mail inputted follows the e-mail of format username@mailserver.domain (the domain could be repeating) Ex: both .edu.lb .gov.edu.lb work.
public void updateUI(FirebaseUser account)	Updates the UI with a Toast message regarding the user's authentication status.
public void onStart()	Override to the onStart() function which checks the user authentication status upon start.
public void OnClickLogIn(View view)	The main method of the Log In class which almost encompasses all the previous methods to log in user in to the VLA system.

The onBackPressed() is useful in restricting the user from navigating back to the activities (windows) to which they may or may not have access to. As such if a user touches on back, nothing will happen as the method overrides the onBackPressed method. Hence, more organization and control in navigation is given to the software designers/developers.

```
1. @Override

public void onBackPressed() {
    // empty so nothing happens
}
```

The OnClickReturnToWelcome() enables the user using intent to go back to the Welcome activity of the VLA system.

```
public void OnClickReturnToWelcome(View view){
    Intent intent = new Intent(this,Welcome.class);
    startActivity(intent);
}
```

The checkInstructorStatus(String email), takes an e-mail string and checks in a regex expression if the e-mail given has .lb at the end. First the regex pattern ".+lb" is sat up and then the matcher method .matches if the String e-mail succeeds in it. Returns true upon correct match, returns false if the e-mail given does not match (instructor).

```
public boolean checkInstructorStatus(String email){
    Pattern p1 = Pattern.compile(".+lb");
    Matcher m1 = p1.matcher(email);

    if (m1.matches()) {
        return true;
    } else {
        return false;
    }

}
```

The checkPassword(String pass), takes a string and checks if it is empty. Returns true if is not empty, false if the password is empty.

```
public boolean checkPassword(String pass) {
    if (pass.isEmpty()) {
        return false;
    }
    return true;
}
```

The validateEmail(String email), takes String as an input, complies a regular expression to verify if the user has input an e-mail in the form username@mailserver.domain(s). Then the matcher method checks if the e-mail string provided matches the format. If it matches (matcher is true) the email_string variable assigns the matched expression between the first matched group in the regular expression ((^\\w+([\\.-]?\\w+)*@\\w+([\\.-]?\\w+)*(\\.\\w{2,3})+$) which translates to the whole e-mail. Hence, the e-mail input becomes assigned to the email_string. After, the method returns true, if no match has been found the method returns false, meaning email is not valid.

```
public boolean validateEmail(String email) {
    //Ensure a valid email is input.
    Pattern p1 = Pattern.compile("(^\\w+([\\.-]?\\w+)*@\\w+([\\.-
]?\\w+)*(\\.\\w{2,3})+$)");
    Matcher m1 = p1.matcher(email_string);

    if (m1.matches()) {
        email_string = m1.group(1);
        return true;
    } else {
        return false;
    }
}
```

The updateUI(FirbaseUser account), takes input account of type FirebaseUser. Used to update status for the user in a dialogue box (Toast) about the user's authentication status. If the account returned a value, a message that the user has been authenticated "You have signed in", else, displays a message "You're not signed in".

```
//Change UI according to user data.
public void updateUI(FirebaseUser account) {
    if (account != null) {
        Toast.makeText(this, "You have signed in",
Toast.LENGTH_LONG).show();
    } else{
        Toast.makeText(this, "You're not signed in",
Toast.LENGTH_LONG).show();
    }
}
```

The onStart(), a void method that on execution checks the authentication status of the user in FirebaseUser variable and passes it to the UpdateUI method.

```
@Override
public void onStart() {
    super.onStart();
    // Check if user is signed in (non-null) and update UI accordingly.
    FirebaseUser currentUser = mAuth.getCurrentUser();
    updateUI(currentUser);
}
```

The OnClickLogIn(View view), takes view in its parameter. A unit in Android development that is used to interact with the User Interface (objects of Android). In this method we will go bit by bit to explain the method clearly.

At first, the method takes the e-mail from the EditText input in the VLA System UI. Then checks if the e-mail is valid by invoking the **validateEmail** method. If the email is invalid, a Toast is shown to the user to input a valid e-mail address and returns (gets out of the method). *return in this method from now on always means gets out of the method.

```
public void OnClickLogIn(View view){

    email_string = input_email.getText().toString();

    //If check e-mail and toast if invalid
    if (!validateEmail(email_string)){
        Toast.makeText(getApplicationContext(), "Please enter a valid
email address.", Toast.LENGTH_SHORT).show();
        dialogue.setText("Please enter a valid email address.");
        return;
    }
```

Then, if e-mail is valid, the password is taken from the VLA UI EditText view, and checks if the password box is empty or not through the **checkPassword** method. If the password is empty, the method displays to the user to input a password and returns.

```
password_string = input_password.getText().toString();

if(!checkPassword(password_string)){
    Toast.makeText(getApplicationContext(), "Please enter a password.",
Toast.LENGTH_SHORT).show();
    dialogue.setText("Please enter a password.");
    return;
}
```

If the user has input a password, then, the method invokes **checkInstructorStatus(email)** and assigns its return value to **isInstructor.** If the checkInstructor has a .lb e-mail address*, then an instructor is trying to sign in, assigning the isInstructor to true. Else, assigning it to false (Student)**

*notice that the ".+lb" will take many values that are do not reflect an e-mail address. However, since we check the e-mail address before putting it in the checkInstructor, then the e-mail will always be valid.

**could be administrator but we did not implement much regarding administrator usage.

```
mAuth.signInWithEmailAndPassword(email_string, password_string)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
{

            @Override
```

Then after getting the status of the user (instructor/student), the method uses the mAuth variable of type FirebaseAuth. With it, we use a method called **.signInWithEmailAndPassword** which takes the previously input email and password strings and sends them to Firebase database for verification. onComplete method then dissects the result through the Task object. If it Task (log in) is successful, the mAuth gets user information and proceeds to intent (Open Activity). If the user is an instructor the VLA System will open the Instructor page. If the user is not an instructor, the VLA will open the Student Homepage.

```
public void onComplete(@NonNull
com.google.android.gms.tasks.Task<AuthResult> task) {
    if (task.isSuccessful()) {
        // Sign in success, update UI with the signed-in user's
information
        Log.d("TAG", "signInWithEmail:success");
        FirebaseUser user = mAuth.getCurrentUser();
        //User is an instructor
        if(isInstructor) {
            Intent intent = new Intent(LogIn.this, Homepage.class);
            startActivity(intent);
        }
        //User is a student
         else if (!isInstructor) {
        Intent intent = new Intent(LogIn.this, HomepageStudent.class);
        startActivity(intent);
         }
```

Meanwhile, if the password and email strings failed to match for a user, the sign in will fail and display authentication failed and then it will update the UpdateUI with the data returned.

```
} else {
    // If sign in fails, display a message to the user.
    Toast.makeText(LogIn.this, "Authentication failed.",
            Toast.LENGTH_SHORT).show();
    updateUI(null);
}
```

2.2 Create Task

The Create Task in the VLA system is based on the Create Task Use-Case previously elaborated on in the document "D&A LAU VLA SRS V1.0". The Create Task can be accessed from the Homepage of an Instructor user by tapping on the "+" button on the bottom right invoking the CreateNewActivity Method. (See Figure 3.0)

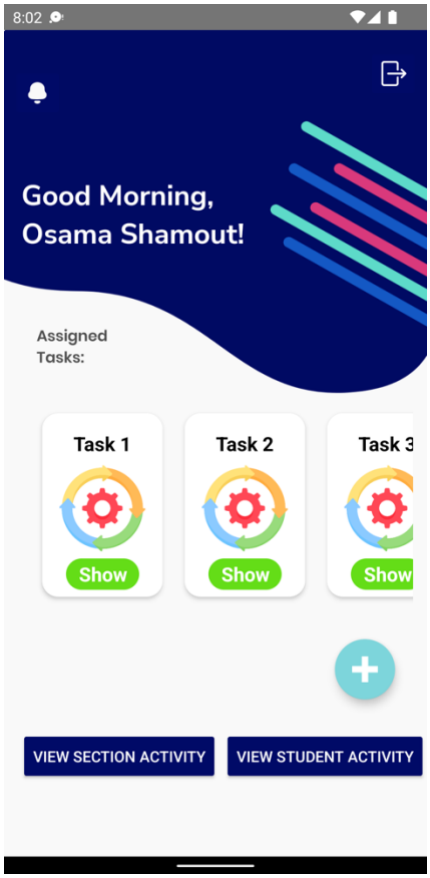


Figure 3: Homepage of Instructor of the VLA System.

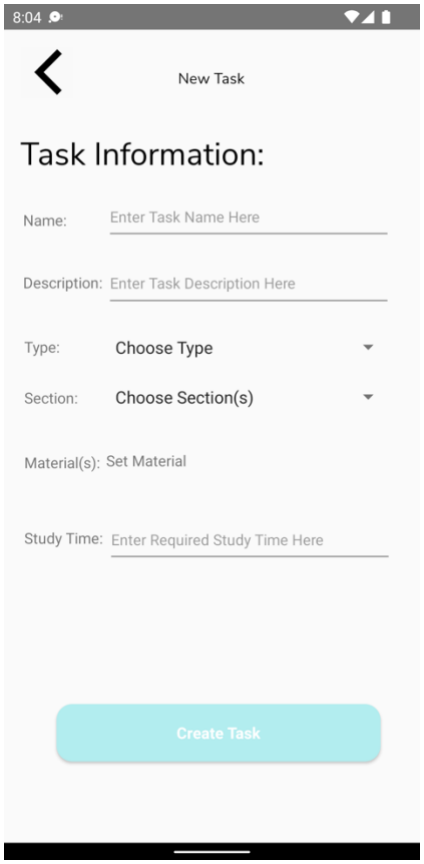


Figure 4: Create Task Page

```
public void OnClickCreateNewActivity(View view){
    Intent intent = new Intent(this,CreateActivity.class);
    startActivity(intent);
}
```

The Homepage will have the variables and methods below:

Variable	Use
String user_id; String name; String[] materialArray float study_hours String task_name_string; String task_description_string; String task_type; String section; String materials; ArrayList<Task> task = new ArrayList<>(); boolean[] selectedMaterial; ArrayList<Integer> materialList = new ArrayList<>();	To retrieve the user session. To retrieve the name of the logged user. Simulates retrieves material. To store the study_hours input from user. To store the task name input from user. To store the task description input from user. To store the task type input from user. To store the task section input from user. To store the task material(s) input from user. An ArrayList to add the tasks created. Boolean array to mark selected material. ArrayList to add the integer of the selected material.
EditText task_name; EditText task_description; EditText time_study; TextView textView; Spinner spinner_list_types; Spinner spinner_list_sections;	Initalize the Views created in the Android Studio task_name to collect input name, task_description to collect input description, time_study to collect input study time, textView to display the materials available.

Methods	Use
onCreate	Intializes the variables and the UI.
public void OnClickReturnHomepage(View view)	Returns to the instructor's homepage.
public boolean checkTaskName(String name)	Checks the task name has been given
public boolean checkTaskDescription(String description)	Checks if task has been given a description
public boolean checkMaterials(String materials)	Updates the UI with a Toast message regarding the user's authentication status.
public boolean checkType(String type)	Override to the onStart() function which checks the user authentication status upon start.
public boolean checkSection(String section)	The main method of the Log In class which almost encompasses all the previous methods to log in user in to the VLA system.

public boolean verifyStudyHours(String hours)	The main method of the Log In class which almost encompasses all the previous methods to log in user in to the VLA system.
public void onClickCreateTask(View view)	The main method of the Log In class which almost encompasses all the previous methods to log in user in to the VLA system.

Before going deeply on the class, I will explain the Spinner. The Spinner class implements onNothingSelected (no overridden variables) and onItemSelected.

```
String text;
public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l)
{
     text = adapterView.getItemAtPosition(i).toString();
}
```

The above code sets the String text to the item value in the selected spinner.

At the onCreate, the spinners and the checklist are initiated.

```
//Define the spinner of types
spinner_list_types= (Spinner) findViewById(R.id.listofTaskTypes);
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
R.array.list_types, android.R.layout.simple_spinner_item);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinner_list_types.setAdapter(adapter);
spinner_list_types.setOnItemSelectedListener(this);
spinner_list_types.setPrompt("Set Task Type:");
```

```
//Define the spinner of sections
spinner_list_sections= (Spinner) findViewById(R.id.listofSections);
ArrayAdapter<CharSequence> adapter2 = ArrayAdapter.createFromResource(this,
R.array.list_sections, android.R.layout.simple_spinner_item);
adapter2.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinner_list_sections.setAdapter(adapter2);
spinner_list_sections.setOnItemSelectedListener(this);
spinner_list_sections.setPrompt("Set Task Section:");
```

Then the material list is created by setting the Boolean variables of the material available and adding to the list the indices of the selected material. Clear all removes all the selected indices, Cancel gets out from the displayed list, ok confirms the selected choices.

```
selectedMaterial = new boolean[materialArray.length];

textView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        // Initialize alert dialog
        AlertDialog.Builder builder = new
```

```
AlertDialog.Builder(CreateActivity.this);

        // set title
        builder.setTitle("Select Material");

        // set dialog non cancelable
        builder.setCancelable(false);

        builder.setMultiChoiceItems(materialArray, selectedMaterial, new
DialogInterface.OnMultiChoiceClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i, boolean
b) {
                // check condition
                if (b) {
                    // when checkbox selected
                    // Add position  in lang list
                    materialList.add(i);
                    // Sort array list
                    Collections.sort(materialList);
                } else {
                    // when checkbox unselected
                    // Remove position from langList
                    materialList.remove(Integer.valueOf(i));
                }
            }
        });

        builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                // Initialize string builder
                StringBuilder stringBuilder = new StringBuilder();
                // use for loop
                for (int j = 0; j < materialList.size(); j++) {
                    // concat array value
                    stringBuilder.append(materialArray[materialList.get(j)]);
                    // check condition
                    if (j != materialList.size() - 1) {
                        // When j value  not equal
                        // to lang list size - 1
                        // add comma
                        stringBuilder.append(", ");
                    }
                }
                // set text on textView
                textView.setText(stringBuilder.toString());
            }
        });

        builder.setNegativeButton("Cancel", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                // dismiss dialog
                dialogInterface.dismiss();
            }
```

```
        });
        builder.setNeutralButton("Clear All", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                // use for loop
                for (int j = 0; j < selectedMaterial.length; j++) {
                    // remove all selection
                    selectedMaterial[j] = false;
                    // clear language list
                    materialList.clear();
                    // clear text view value
                    textView.setText("");
                }
            }
        });
        // show dialog
        builder.show();
    }
});
```

The OnClickReturnHomepage(View view), returns to the previous page upon being called by the back button (arrow to left).

```
public void OnClickReturnHomepage(View view){
    Intent intent = new Intent(this,Homepage.class);
    startActivity(intent);
}
```

The checkTaskName(String name), takes the String name and check if it is empty and contains a valid input. The method compiles the regex at the Patter p1 for any combination of words and removes the empty spaces at the beginning of the line when grouping in m1.matches → m1.group(2) as number 2 means the second caught group. If the task name was given correctly, returns true, else returns false.

```
public boolean checkTaskName(String name){
    //Ensure a task name is given.
    Pattern p1 = Pattern.compile("^(\\s?)+((.)+)");
    Matcher m1 = p1.matcher(name);

    if(m1.matches()) {
        task_name_string = m1.group(2);
        return true;
    }else{
        return false;
    }
}
```

The checkTaskDescription, identical in its execution behavior of the checkTaskName, returns true if a task_description has been given, returns false if no valid description was given.

```
public boolean checkTaskDescription(String description){
    //Ensure a task name is given.
    Pattern p1 = Pattern.compile("^(\\s?)+((.)+)");
    Matcher m1 = p1.matcher(description);

    if(m1.matches()) {
        task_description_string = m1.group(2);
        return true;
    }else{
        return false;
    }
}
```

The checkMaterials(String materials), takes the String material and checks if the user has input an empty material list (null or empty string “”) or left the input at the “Set Material”. Returns false if any of the previous indicator of empty material is given, returns true if material was chosen.

```
public boolean checkType(String type){
    if(type.isEmpty() || type.equals("") || type.equalsIgnoreCase("Choose
Type")){
        return false;
    }
    return true;
}
```

The checkSection(String section), identical in its execution behavior to checkMaterials. Returns false if section is empty, true if section is input.

```
public boolean checkMaterials(String materials){
    if(materials.isEmpty() || materials.equals("") ||
materials.equalsIgnoreCase("Set Material")){
        return false;
    }
    return true;
}
```

The checkType(String type), identical in its execution behavior to checkMaterials. Returns false if type is empty, true if type is input.

```
public boolean checkType(String type){
    if(type.isEmpty() || type.equals("") || type.equalsIgnoreCase("Choose
Type")){
        return false;
    }
    return true;
}
```

The verifyStudyHours(String hours), takes a String hours, converts it into a float value (in case the study hours have fraction) and assigns it to study_hours. The catch clause catches the Exception in case user enters invalid value (InputMismatch etc.). and displays input upon occurring. Then, it returns false. If no exception was raised, the method moves to check if the study_hours are above 0. If it fails (study_hours<=0), value returns false (study hours not valid). If it passes all the previous checks, the method returns true (valid study_hours).

```
public boolean verifyStudyHours(String hours){
    try{
        study_hours = Float.parseFloat(time_study.getText().toString());
    }
    catch (Exception e){
        Toast.makeText(getApplicationContext(), "Invalid Data Input",
Toast.LENGTH_SHORT).show();
        return false;
    }
    if (!(study_hours >0)){
        return false;
    }
    return true;
}
```

The onClickCreateTask, the pivotal method in the class creates task an instantiates the Task class to add a new task. First, we obtain the task name and task description from user and assign them to their respective Strings

```
public void onClickCreateTask(View view){

    //Obtain all information to register activity
    task_name_string = task_name.getText().toString();
    task_description_string = task_description.getText().toString();
```

Second, we start to obtain and validate data, step by step. We check for the task name by invoking the checkTaskName and giving it a task_name_string in its parameters. If the checkTaskName fails it displays to the user to input a valid task name. Similarly the checkTaskDescription functions. At the end of both if statements, a return in case they did not succeed in checking.

```
if(!checkTaskName(task_name_string)){
    Toast.makeText(getApplicationContext(), "Please enter a valid task name",
Toast.LENGTH_SHORT).show();
    return;
}

if(!checkTaskDescription(task_description_string)){
    Toast.makeText(getApplicationContext(), "Please enter a valid task
description", Toast.LENGTH_SHORT).show();
    return;
}
```

Then, we move to obtain the Task Type. This is done from the Spinner class by using the .getSelectedItemPosition() which gives the index of the item chosen, then, it gets translated to a string through the use of getItemAtPosition(number).toString().

```
//Obtain Task Type
AdapterView<?> parent = spinner_list_types;
int number = spinner_list_types.getSelectedItemPosition();
task_type = spinner_list_types.getItemAtPosition(number).toString();
```

After, we check if the type input is correct. If invalid a toast is displayed to user and the method returns.

```
if(!checkType(task_type)){
    Toast.makeText(getApplicationContext(), "Invalid Task Type Input",
Toast.LENGTH_SHORT).show();
    return;
}
```

We repeat the same steps of the Task Type to Task Section.

```
//Obtain Task Section
AdapterView<?> parent2 = spinner_list_sections;
int number2 = spinner_list_sections.getSelectedItemPosition();
section = spinner_list_sections.getItemAtPosition(number2).toString();
```

Then we check for the input section if it is correct. If invalid a toast is displayed to user and the method returns.

```
if(!checkSection(section)){
    Toast.makeText(getApplicationContext(), "Invalid Section Input",
Toast.LENGTH_SHORT).show();
    return;
}
```

Then we take the material from the list built from the onCreate method (the choices the user has chosen from the menu available)

```
materials = textView.getText().toString();
```

Then we check if the materials are valid (not empty). If invalid, the method returns.

```
if(!checkMaterials(materials)){
    Toast.makeText(getApplicationContext(), "Invalid Material Input",
Toast.LENGTH_SHORT).show();
    return;
}
```

Finally the last piece of information needed for the task is the Study Hours. Checked in an if statement. If the time_study.getText().toString() (the value input by user) is not valid, display to user the condition that must be satisfied and return.

```
//Obtain Study Hours Section and check condition
if(!(verifyStudyHours(time_study.getText().toString()))){
    //If study hours less than 0
    Toast.makeText(getApplicationContext(), "Hours must be greater than 0.",
Toast.LENGTH_SHORT).show();
    return;
}
```

If the onClickCreateTask reached this stage, then all the information is valid and now we can add the task to the ArrayList available at the Homepage (Instructor) and Homepage_student(Student). Hence, we need to instantiate both of these classes and then instantiate an instance of the Task class and fill it with the variables previously obtained. At the status variable, we input “Incomplete” as all newly created tasks have the status of incomplete. Afterwards, we call the method getTask() from the Homepage and Homepage_student, then we append to the ArrayList in both classes the newly created task instance. Finally, we display via Toast to user that the Task has been successfully created.

```
Homepage page = new Homepage();
HomepageStudent page2 = new HomepageStudent();

Task task1 = new Task(task_name_string,task_type, section, study_hours,
"Incomplete", materials);

//Log.e("Task added is:", task1.toString());

page.getTask().add(task1);
page2.getTask().add(task1);

Toast.makeText(getApplicationContext(), "Task Added Successfully",
Toast.LENGTH_SHORT).show();
```

3. Unit Testing

3.1 Log in

```
boolean result = validateEmail(email_string);

if (!result){
    Toast.makeText(getApplicationContext(), "Please enter a valid email address.",
Toast.LENGTH_SHORT).show();
    dialogue.setText("Please enter a valid email address.");
    return;
}
```

Method	On Login
Input	Email equals null
Expected Output	false
Actual Output	false
Verified?	Yes

E/Result: false

Method	On Login
Input	"osama”
Expected Output	false
Actual Output	false
Verified?	Yes

E/Result: false

Method	On Login
Input	"osama.shamout@lau.edu”
Expected Output	true
Actual Output	true
Verified?	Yes

E/Result: true

Method	On Login
Input	"omar.mlaeb@lau.edu.lb”
Expected Output	true
Actual Output	true
Verified?	Yes

E/Result: true

```
if(!checkPassword(password_string)){
    Toast.makeText(getApplicationContext(), "Please enter a password.",
Toast.LENGTH_SHORT).show();
    dialogue.setText("Please enter a password.");
```

```
    return;
}
```

Method	On Login
Input	Password equals null
Expected Output	false
Actual Output	false
Verified?	Yes

E/Result: false

Method	On Login
Input	"123123_Aa”
Expected Output	true
Actual Output	true
Verified?	Yes

E/Result: true

Method	On Login
Input	"12311123_Aaa”
Expected Output	true
Actual Output	true
Verified?	Yes

E/Result: true

3.2 Create Task

1. CheckTaskName:

```
boolean name_check = checkTaskName(task_name_string);
Log.e("return", String.valueOf(name_check));
```

Figure 5

Method	checkTaskName
Input	“”
Expected Output	false
Actual Output	false
Verified?	Yes

2022-05-16 23:42:29.631 16452-16452/com.example.studypal E/return: false

Figure 6

Method	checkTaskName
Input	“Task 5”
Expected Output	true
Actual Output	true
Verified?	Yes

2022-05-16 23:51:37.822 16452-16452/com.example.studypal E/return: true

Figure 7

Method	checkTaskName
Input	123
Expected Output	true
Actual Output	true
Verified?	Yes

2022-05-16 23:56:47.200 16452-16452/com.example.studypal E/return: true

2. CheckTaskDescription

```
boolean description_check = checkTaskDescription(task_description_string)
if(!description_check){
    Log.e("return", String.valueOf(description_check));
    Toast.makeText(getApplicationContext(), "Please enter a valid task
description", Toast.LENGTH_SHORT).show();
    return;
}
```

Figure 8

Method	checkTaskDescription
Input	“”
Expected Output	false
Actual Output	false
Verified?	Yes

2022-05-16 23:51:46.395 16452-16452/com.example.studypal E/return: false

Figure 9

Method	checkTaskDescription
Input	“This is a test”
Expected Output	true
Actual Output	true
Verified?	Yes

2022-05-16 23:54:37.822 16452-16452/com.example.studypal E/return: true

Figure 10

Method	checkTaskDescription
Input	123123
Expected Output	true
Actual Output	true
Verified?	Yes

2022-05-17 00:10:17.537 16452-16452/com.example.studypal E/return: true

3. CheckTaskType

```
boolean type_check = checkType(task_type);
Log.e("Result", String.valueOf(type_check));
if(!type_check){
    Toast.makeText(getApplicationContext(), "Invalid Task Type Input",
Toast.LENGTH_SHORT).show();
    return;
}
```

Figure 11

Method	checkTaskType
Input	“”

Expected Output	false
Actual Output	false
Verified?	Yes

2022-05-17 00:16:22.636 16452-16452/com.example.studypal E/Result: false

Figure 12

Method	checkTaskType
Input	“Choose a Type”
Expected Output	False
Actual Output	False
Verified?	Yes

2022-05-17 00:21:22.636 16452-16452/com.example.studypal E/Result: false

Figure 13

Method	checkTaskType
Input	“Material”
Expected Output	True
Actual Output	True
Verified?	Yes

2022-05-17 00:22:59.306 16452-16452/com.example.studypal E/Result: true

Figure 14

Method	checkTaskType
Input	“Exam”
Expected Output	True
Actual Output	True
Verified?	Yes

2022-05-17 00:25:17.741 16452-16452/com.example.studypal E/Result: true

4. CheckTaskSection

```
boolean section_check = checkSection(section);
Log.e("Result", String.valueOf(section_check));
if(!section_check){
    Toast.makeText(getApplicationContext(), "Invalid Section Input",
Toast.LENGTH_SHORT).show();
    return;
}
```

Figure 15

Method	checkTaskSection
Input	“Choose Section(s)”
Expected Output	False
Actual Output	False
Verified?	Yes

2022-05-17 00:30:16.051 16452-16452/com.example.studypal E/Result: false

Figure 16

Method	checkTaskSection
Input	Section 11
Expected Output	True
Actual Output	True
Verified?	Yes

2022-05-17 00:31:03.522 16452-16452/com.example.studypal E/Result: true

Figure 17

Method	checkTaskSection
Input	Section 12
Expected Output	True
Actual Output	True
Verified?	Yes

2022-05-17 00:33:30.321 16452-16452/com.example.studypal E/Result: true

5. CheckTaskMaterial

```
boolean material_check = checkMaterials(materials);
Log.e("Result", String.valueOf(material_check));
if(!material_check){
    Toast.makeText(getApplicationContext(), "Invalid Material Input",
Toast.LENGTH_SHORT).show();
    return;
}
```

Figure 18

Method	checkTaskMaterial
Input	“Set Material”
Expected Output	False
Actual Output	False
Verified?	Yes

2022-05-17 00:38:35.607 16452-16452/com.example.studypal E/Result: true

Figure 19

Method	checkTaskMaterial
Input	“Chapter 1, Chapter 3, Chapter 4”
Expected Output	True
Actual Output	True
Verified?	Yes

2022-05-17 00:31:03.522 16452-16452/com.example.studypal E/Result: true

Figure 20

Method	checkTaskMaterial
Input	“”
Expected Output	False
Actual Output	False
Verified?	Yes

2022-05-17 00:40:30.379 16452-16452/com.example.studypal E/Result: false

6. verifyStudyHours

```
boolean check_hours = verifyStudyHours(time_study.getText().toString());
Log.e("Result", String.valueOf(check_hours));
if(!check_hours){
    //If study hours less than 0
    Toast.makeText(getApplicationContext(), "Hours must be greater than 0.",
Toast.LENGTH_SHORT).show();
    return;
}
```

Figure 21

Method	verifyStudyHours
Input	256.5233921
Expected Output	true
Actual Output	true
Verified?	Yes

2022-05-17 00:46:16.951 16452-16452/com.example.studypal E/Result: true

Figure 22

Method	verifyStudyHours
Input	699.
Expected Output	True
Actual Output	True
Verified?	Yes

2022-05-17 00:47:14.168 16452-16452/com.example.studypal E/Result: true

Figure 23

Method	verifyStudyHours
Input	.
Expected Output	False
Actual Output	False
Verified?	Yes

2022-05-17 00:47:59.206 16452-16452/com.example.studypal E/Result: false

Figure 24

Method	verifyStudyHours
Input	“”
Expected Output	False
Actual Output	False
Verified?	Yes

2022-05-17 00:41:24.258 16452-16452/com.example.studypal E/Result: false

**Note: the Android input keyboard has been restricted to decimals hence checking for negative values or values other than numbers is not possible.

4. System Testing

4.1 Log In as a Use-Case

```
if (task.isSuccessful()) {
    // Sign in success, update UI with the signed-in user's information
    Log.d("TAG", "signInWithEmail:success");
    FirebaseUser user = mAuth.getCurrentUser();
    //User is an instructor
    if(isInstructor) {
        Intent intent = new Intent(LogIn.this, Homepage.class);
        startActivity(intent);
    }
    //User is a student
     else if (!isInstructor) {
    Intent intent = new Intent(LogIn.this, HomepageStudent.class);
    startActivity(intent);
     }
} else {
    // If sign in fails, display a message to the user.
    Toast.makeText(LogIn.this, "Authentication failed.",
            Toast.LENGTH_SHORT).show();
    updateUI(null);
}
```

Method	On Complete
Input	“osama.shamout@lau.edu.lb” , “123123_Aa”
Expected Output	true
Actual Output	true
Verified?	Yes

E/Result: true

Method	On Complete
Input	“osama.shamoutlau.edu.lb” , “123123_Aa”
Expected Output	false
Actual Output	false
Verified?	Yes

E/Result: false

Method	On Complete
Input	“osama.shamout@lau.edu.lb” , “12312a”
Expected Output	false
Actual Output	false
Verified?	Yes

E/Result: false

Method	On Complete

Input	“osama.samout@lau.du.lb” , “123123a”
Expected Output	false
Actual Output	false
Verified?	Yes

E/Result: false

4.2 Create Task as a Use-Case

onClickCreateTask

```
public void onClickCreateTask(View view){

//…
//.. Code tested as units
//..
    Homepage page = new Homepage();
    HomepageStudent page2 = new HomepageStudent();

    Task task1 = new Task(task_name_string,task_type, section, study_hours,
"Incomplete", materials, "work_process_img");

    Log.e("Task added is:", task1.toString());
try{
    page.getTask().add(task1);
    page2.getTask().add(task1);
}
catch(Exception e){
    Toast.makeText(getApplicationContext(), "Could not add task.",
Toast.LENGTH_SHORT).show();
Return;
}
Toast.makeText(getApplicationContext(), "Task Added Successfully",
Toast.LENGTH_SHORT).show();
```

Method	Task constructor
Input	Task1, This is a test, Section 11, 12, "Incomplete", “Chapter 2, Chapter 3, Chapter 4”, "work_process_img")
Expected Output	Task Added Successfully Dialogue
Actual Output	Task Added Successfully Dialogue
Verified?	Yes

E/Task added is:: Task{name='Task 1', type='Material', section='Section 11', img='work_process_img', progress=0, study_hours=12.0, status='Incomplete', materials='Chapter 2, Chapter 3, Chapter 4'}

Dialog box Task Added Successfully Displayed. See Figures in Appendex.
No other cases could be made in this Use-Case as the the data is validated step by step. Hence, no odd information could be input in the Task instance.

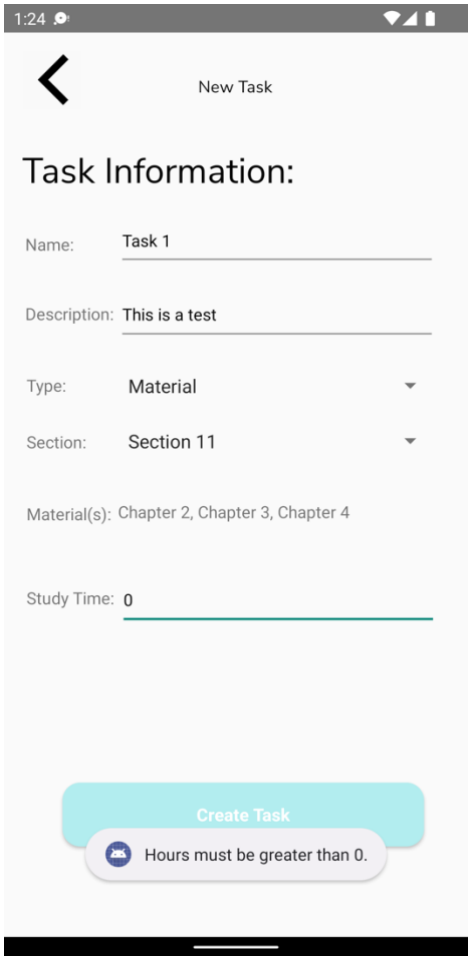
Method	.add
Input	(Task1) object
Expected Output	Task Added Successfully Dialogue

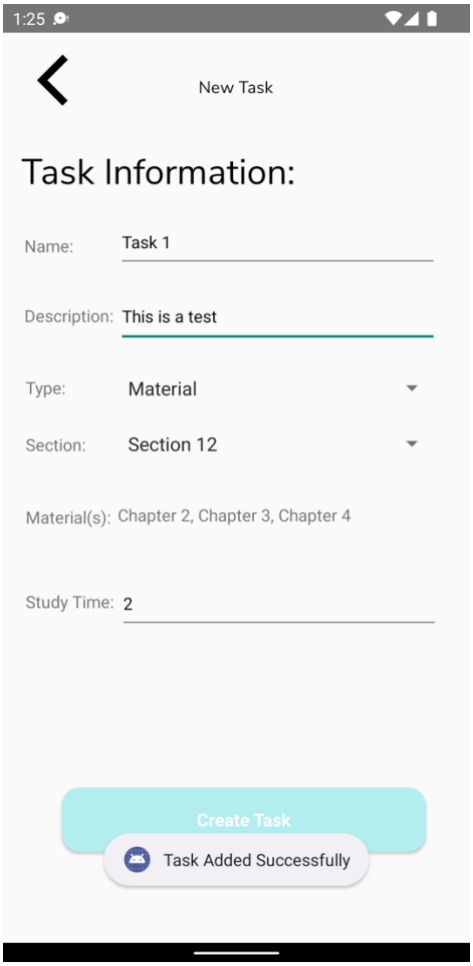
Actual Output	Task Added Successfully Dialogue
Verified?	Yes

*If the task was not added an exception would have arisen. Yet since we cannot have values that are invalid at this stage, there is no case where the input could give anomalies.

Overall steps:

1. User inputs task name.
2. User inputs task description.
3. User inputs task type.
4. User inputs task section.
5. User inputs task material(s).



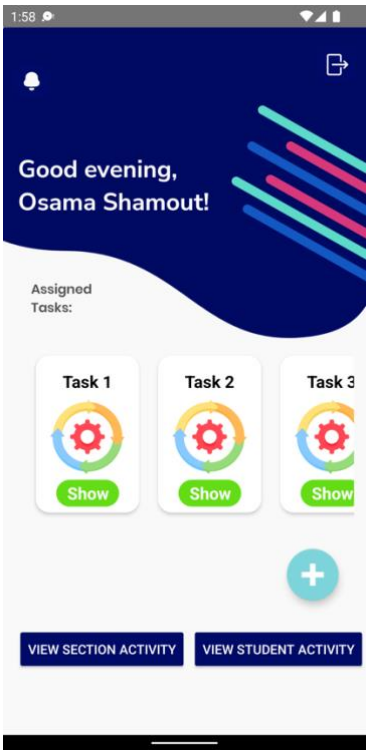


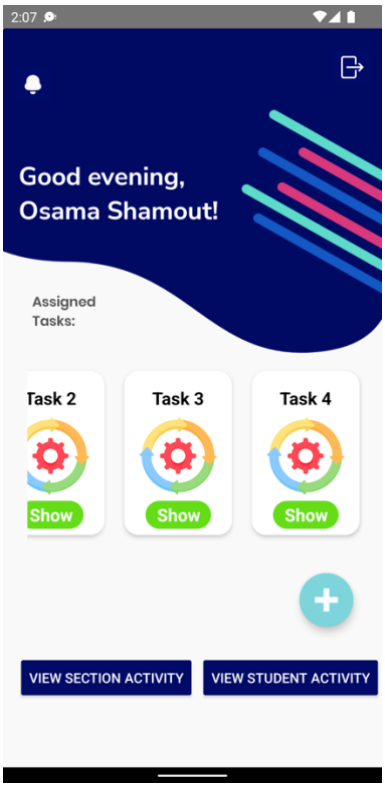
6. User inputs task study time.

System checks if any of the above conditions is invalid. If invalid, it returns a dialogue of the first invalid input. If all valid. New task is created successfully.

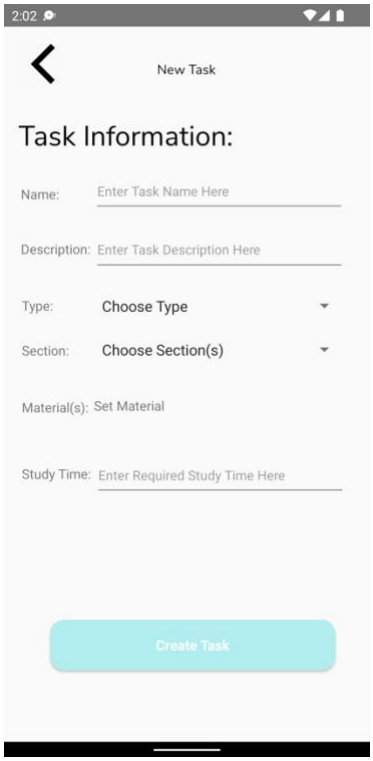
4.3 Whole System

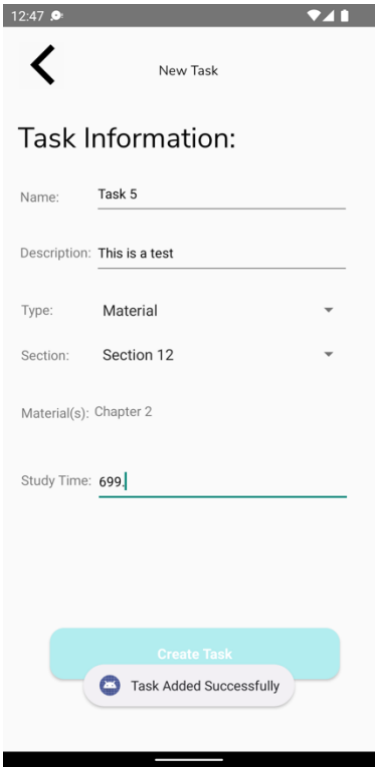
Inputs: email, password
osama.shamout@lau.edu.lbosama.shamout@lau.edu.lb 123123_Aa → System Log In

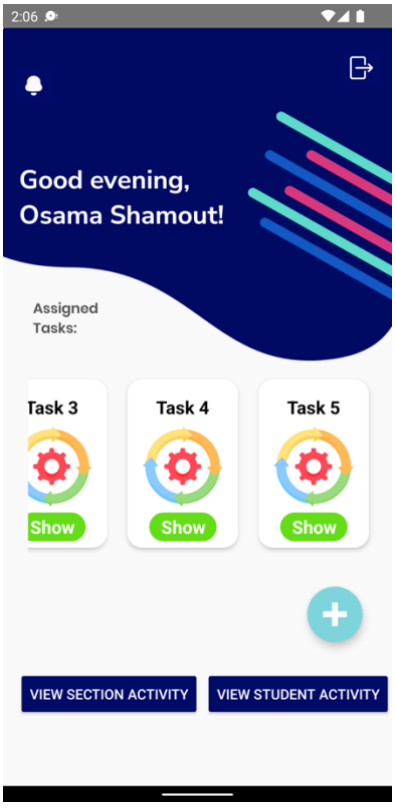




Inputs: name, description, type, section, material(s), study time.
Output: create task.







4.3 NFREQ Testing for Response Time

The most complex transaction is Create Task. Hence, I will be simulating the time needed to create a task between the time it takes the user to input all the date and the time they tap on Create Task. Hence, we will modify the code accordingly to reflect create a time interval. Start becomes the first line after the user touches the onClickCreateTask() and the End is after the line containing "Task Added Successfully".

```
public void onClickCreateTask(View view){
    long start = System.currentTimeMillis();
```

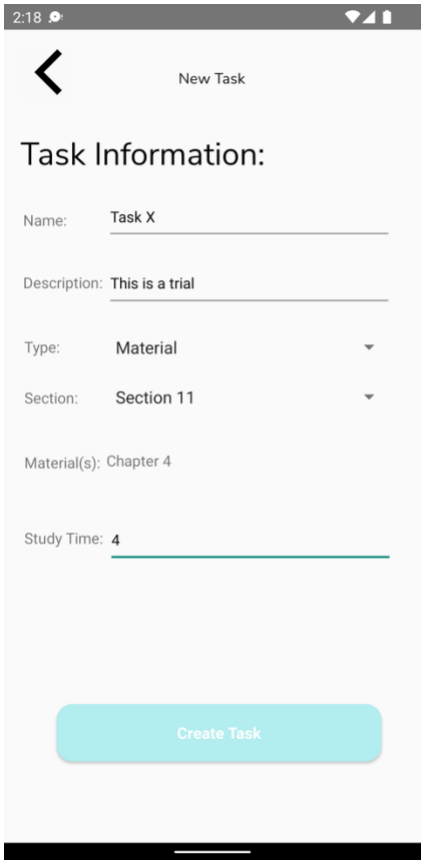
The code in between is here.

```
  Toast.makeText(getApplicationContext(), "Task Added Successfully",
Toast.LENGTH_SHORT).show();
        long finish = System.currentTimeMillis();

        long timeElapsed = finish - start;

        Log.e("Time to create task:", String.valueOf(timeElapsed) + "
millisecond");
    }
}
```

The time elapsed to registered the activity to the right:

2022-05-17 02:18:23.167 17907-17907/com.example.studypal E/Task added is:: Task{name='Task X', type='Material', section='Section 11', img='work_process_img', progress=0, study_hours=4.0, status='Incomplete', materials='Chapter 4'}

2022-05-17 02:18:23.170 17907-17907/com.example.studypal E/Time to create task in ms:: 9 millisecond



4.1.4 NFREQ-4: The system shall have a response time of less than 4000 milliseconds for all database transactions and/or processes being performed at the 95th percentile (modified for more realistic time)

9 < 4000

Hence, the system satisfies the NFREQ-4.

It is sure to note that since small amount of data and minimal DB APIs are used. It certainly will bring down the response time significantly as data is accessed locally.

5. Conclusion