



**Goal Tracking App Project Report**

**Executive Summary**

This comprehensive report details the strategic approach to the development and design of a sophisticated Goal Tracking App. The app is specifically designed to empower individuals in efficiently managing and accomplishing their personal and professional objectives. It offers a suite of features, encompassing goal setting, progress tracking, data visualization, and seamless integration with external productivity tools.

**Introduction and Motivation**

The "Goal Tracker" app is developed to assist users in managing their tasks and goals efficiently. The app provides a user-friendly interface to create, update, and track various tasks with priorities, start and end dates, and descriptions. The motivation behind this project is to offer users a digital tool that helps them organize their tasks, set priorities, and enhance their overall productivity.

**Problem Statement**

Managing tasks and goals can become challenging in today's fast-paced world. Traditional methods of task management, such as physical to-do lists, may lack flexibility and organization. The Goal Tracker app aims to address this problem by providing users with a digital platform to manage and track their tasks more effectively.

**Project Overview**

The Goal Tracking App stands as an indispensable asset for individuals seeking enhanced organization and productivity in their pursuit of goals. The app is enriched with a spectrum of transformative features:

* Goal Setting: Users can meticulously articulate their objectives, detailing title, description, deadline, and priority level, thereby enabling comprehensive and effective planning.
* Progress Tracking: The app equips users to log activities, delineate milestones, and append notes to their goals. Gentle reminders ensure unwavering focus and unwavering commitment.
* Visualization: A captivating dashboard, enriched with interactive charts and graphs, offers users an instant, real-time overview of their progress, translating achievements into visually engaging representations.

**Scope**

This project spans a broad scope, encompassing an array of cutting-edge features:

* Integration with Productivity Tools: Seamlessly dovetailing with established productivity tools like calendars, to-do lists, and project management platforms, the app harmonizes task organization and prioritization.
* Data Security and Privacy Measures: Imposing stringent data security protocols, including encryption, authentication, and data fortification mechanisms, ensures the utmost confidentiality of user information.

**Key Focus Areas**

The project centres its attention on the following pivotal domains:

* Implementing Progress Bar: A dynamic, visually illuminating progress bar harmoniously integrated into the user interface, delivering real-time updates on ongoing tasks and processes.
* Google Login Integration: By orchestrating seamless authentication via Google Login, the user experience attains unprecedented levels of convenience and accessibility.
* Enhancing Note Deletion: The introduction of user confirmation prompts during note deletion fortifies the app's user-centricity, mitigating the potential for inadvertent data loss.
* Elevating UI Aesthetics: The user interface design undergoes meticulous refinement, encompassing colour palettes, typography, spacing, and navigation. The outcome is a refined, professional, and user-centric design language.
* Tailored Feature Addition: By discerning and addressing project-specific requisites, the app will accommodate additional functionalities that synchronize with user productivity, data management, and collaborative needs.
* Optimizing Database Structure: Rigorous assessment and refinement of the database structure ensure congruence with evolving project requisites and benchmarks for performance excellence.

**Integration with Productivity Tools**

The app's profound integration with established productivity tools, ranging from calendars to to-do lists and project management systems, empowers users to navigate their tasks seamlessly, amplifying overall efficiency.

**Graph Library and Database Implementation**

Elevating user experience, a state-of-the-art graph library furnishes dynamic visualizations, enhancing data comprehension. The incorporation of an optimized SQLite database structure underscores data efficiency and uncompromised access.

**SplashScreenActivity.java:**

1. onCreate(Bundle savedInstanceState):
   1. This method is the entry point for the activity's lifecycle.
   2. It initializes the activity and sets up the user interface with the content view defined in activity\_splash\_screen.xml.
   3. Inside this method, a Handler is used to delay the execution of code by 3000 milliseconds (3 seconds), creating a splash screen effect.
   4. The postDelayed method of the Handler is used to run a piece of code after the specified delay.
2. run() (Inside postDelayed Runnable):
   1. This method contains the code that will be executed after the specified delay.
   2. It creates an explicit Intent to transition from the SplashScreenActivity to the LoginActivity.
   3. The startActivity method is called to launch the LoginActivity.
   4. After launching the new activity, the finish method is called on the current SplashScreenActivity to close it and remove it from the back stack.
   5. The SplashScreenActivity class is responsible for creating a splash screen that is displayed when the app starts. The splash screen is shown for 3 seconds (as specified by the delay in the Handler), and during this time, it transitions to the LoginActivity. This class serves as a visual introduction to the app and provides a smooth transition to the main part of the application.

**LoginActivity.java:**

1. onCreate(Bundle savedInstanceState): Initializes the activity, sets up UI elements, and handles click events for login, forgot password, and sign up actions.
2. saveLoginCredentials(String name, String inputEmail, String inputPassword): Saves login credentials (name, email, password) in SharedPreferences.
3. onClick(View view): Handles the login button click event, validates input fields, checks login credentials against the database, and navigates to the main activity upon successful login.
4. onClick(View view): Handles the forgot password text click event, navigates to the ForgetPasswordActivity.
5. onClick(View view): Handles the sign-up text click event, navigates to the SignupActivity.

**SignupActivity.java:**

1. onCreate(Bundle savedInstanceState): Initializes the activity, sets up UI elements, and handles the sign-up button click event.
2. onClick(View view): Handles the sign-up button click event, validates input fields (name, email, password), saves user signup data in the database, and navigates to the LoginActivity upon successful signup.
3. onClick(View view): Handles the login text click event, navigates to the LoginActivity.
4. saveSignupData(String name, String email, String password): Saves user signup data (name, email, password) in the database.

**ForgetPasswordActivity.java:**

1. onCreate(Bundle savedInstanceState): Initializes the activity, sets up UI elements, and handles the reset button click event.
2. onClick(View v): Handles the reset button click event, validates input fields (email, passwords), performs password reset if conditions are met, and navigates back to the LoginActivity.
3. onBackPressed(): Handles the back button press, finishes the activity.

**NavigationDrawerActivity.java:**

1. onCreate(Bundle savedInstanceState):
   1. This method is the entry point for the activity's lifecycle.
   2. It initializes the activity and sets up the user interface.
   3. The navigation drawer layout, toolbar, and bottom navigation view are set up in this method.
   4. If the user is not logged in (based on SharedPreferences), they are redirected to the login screen.
   5. If the user is logged in, their profile information is displayed in the navigation drawer's header.
2. onSupportNavigateUp():
   1. This method is called when the user presses the "Up" button in the toolbar.
   2. It's used to navigate up within the app's navigation hierarchy.
   3. It utilizes the NavController to navigate up, either within the current fragment or to a parent fragment.
3. Profile Image Click Listener:
   1. The profile image within the navigation drawer has an OnClickListener set.
   2. When clicked, it launches the ProfileActivity where the user can view and edit their profile information.
4. BottomNavigationView Setup:
   1. The BottomNavigationView is initialized and set up to work with the NavController.
   2. The setupWithNavController method is used to ensure that the selected item in the bottom navigation bar corresponds to the current destination in the navigation graph.
5. NavigationUI Setup:
   1. The NavigationUI class is used to connect various UI components to the navigation controller.
   2. The setupActionBarWithNavController method connects the toolbar with the navigation controller, enabling proper navigation behavior when the "Up" button is pressed.
   3. The setupWithNavController method connects the navigation drawer and the bottom navigation view with the navigation controller, ensuring proper navigation behavior when items are selected.
6. SharedPreferences Usage:
   1. The activity uses SharedPreferences to check if the user is logged in.
   2. If the user is not logged in, they are redirected to the login screen using an explicit Intent.
   3. The NavigationDrawerActivity serves as the main hub for navigating through different sections of the app. It ensures that the navigation drawer, toolbar, and bottom navigation view work seamlessly together, providing users with an intuitive and consistent navigation experience. By connecting various UI elements to the NavController, it enables navigation between fragments and keeps the user informed about their current location within the app. The activity also manages user authentication through SharedPreferences, ensuring that only authenticated users can access the app's features.

**HomeFragment:**

1. onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState):
   1. Inflates the layout for the HomeFragment and initializes UI elements.
   2. Sets up the welcome message for the logged-in user.
2. onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState):
   1. Initializes the **RecyclerView**, sets up the **TaskAdapter**, and attaches a **SwipeToDeleteCallback** for task deletion.
3. onAttach(@NonNull Context context):
   1. Attaches the context to the fragment.
4. onDetach():
   1. Detaches the context from the fragment.

**GoalMonitoringFragment:**

1. onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState):
   1. Inflates the layout for the GoalMonitoringFragment and initializes UI elements like TextView and PieChart.
2. getData():
   1. Retrieves task data from the database and calculates the count and percentage of tasks for each priority level.
3. setData():
   1. Displays the calculated priority percentages in a pie chart.
4. onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState):
   1. Initializes the fragment, retrieves data, and sets up the pie chart visualization.
5. onAttach(@NonNull Context context):
   1. Attaches the context to the fragment.
6. onDetach():
   1. Detaches the context from the fragment.

**AddGoalsFragment:**

1. onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState):
   1. Inflates the layout for the AddGoalsFragment and initializes UI elements.
   2. Sets up the spinners for priority selection and date pickers for task dates.
2. initView(View view):
   1. Initializes UI elements.
3. AddTaskEvent():
   1. Defines behavior when the "Add Goal" button is clicked.
   2. Validates input fields, inserts a new task into the database, and simulates loading before navigating to the main activity.
4. onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState):
   1. Initializes the fragment, sets up the database connection.
5. onAttach(@NonNull Context context):
   1. Attaches the context to the fragment.
6. onDetach():
   1. Detaches the context from the fragment.

**ContactUsFragment:**

1. onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState):
   1. Inflates the layout for the ContactUsFragment and initializes UI elements.
2. initView(View view):
   1. Initializes UI elements.
3. submitContactUsFormEvent():
   1. Defines behavior when the "Submit" button is clicked.
   2. Validates user input, displays a loading animation, simulates form submission, and resets the UI.

**SettingFragment:**

1. onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState):
   1. Inflates the layout for the SettingFragment and sets up the click listener for the "SignOut" TextView.
2. clearLoginCredentials():
   1. Clears the saved login credentials from SharedPreferences.

These methods are crucial in defining the functionality of each fragment in your Goal Tracking App, from displaying data to handling user interactions and managing transitions between different app states.

**ProfileActivity.java:**

1. onCreate(Bundle savedInstanceState):
   1. This method is the entry point for the activity's lifecycle.
   2. It initializes the activity and sets up the user interface with the content view defined in activity\_profile.xml.
   3. It initializes various views like userNameTextView, userEmailTextView, etc.
   4. Calls the fetchAndDisplayUserDetails() method to retrieve saved user details from SharedPreferences and update the UI.
2. fetchAndDisplayUserDetails():
   1. This method retrieves saved user details (name, email, password) from the SharedPreferences.
   2. Updates the UI elements (userNameTitleTextView, userNameTextView, userEmailTextView, userPasswordTextView) with the retrieved data.

Certainly, here's a list of methods used in each class along with their descriptions:

**SingleGoalActivity.java**:

1. onCreate(Bundle savedInstanceState):
   1. This method initializes the activity and sets up the user interface with the content view defined in `activity\_single\_goal.xml`.
   2. It initializes various views like `goalNameTextView`, `goalPriorityTextView`, etc.
   3. Retrieves task details from shared preferences and updates the UI with those details.
   4. Calls the `wantToUpdateGoal()` method to set up an onClickListener for the "Update Goal" button.
2. wantToUpdateGoal():
   1. This method sets an onClickListener for the "Update Goal" button (`btnUpdateGoal`).
   2. When the button is clicked, it navigates to the `UpdateTaskActivity` using an explicit `Intent`.

**UpdateTaskActivity.java:**

1. onCreate(Bundle savedInstanceState):
   1. This method initializes the activity and sets up the user interface with the content view defined in `fragment\_add\_goals.xml`.
   2. Initializes various views and fields (`task\_Name`, `spinner`, etc.).
   3. Retrieves task details from shared preferences and sets the values of corresponding views.
   4. Sets up the spinner for selecting priority levels and sets a selected item if the task has a priority set.
   5. Calls the `UpdateTaskEvent()` method to set up an onClickListener for the "Update Goal" button.
2. UpdateTaskEvent():
   1. This method sets an onClickListener for the "Update Goal" button (`btnAddGoal`).
   2. Then the button is clicked, it calls the `updateTask()` method of the `MyDatabaseHelper` class to update the task details in the database.
   3. Displays a toast message indicating the item was updated.
   4. Navigates to the `NavigationDrawerActivity` using an explicit `Intent`.

**MyDatabaseHelper.java:**

1. onCreate(SQLiteDatabase db):
   * This method is called when the database is created for the first time.
   * It executes SQL statements to create the necessary tables for tasks and login data.
2. onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion):
   * This method is called when the database needs to be upgraded.
   * It drops the existing tables and recreates them with the updated structure.
3. checkSignInDetails(String email, String password):
   * This method queries the login table to check if a user with the given email and password exists.
   * Returns a **Cursor** with the user details if found, otherwise returns **null**.
4. insertTask(Task task):
   * This method inserts a new task into the tasks table.
   * Returns the ID of the newly inserted row.
5. updateTask(SQLiteDatabase db, int id, String name, String priority, String startDate, String endDate, String endDescription):
   * This method updates an existing task in the tasks table based on the provided ID.
6. deleteTask(int id):
   * This method deletes a task from the tasks table based on the provided ID.
   * Returns the number of rows affected.
7. getAllTasks():
   * This method retrieves all tasks from the tasks table.
   * Returns an **ArrayList<Task>** containing all tasks.

**LoginContract.java:**

* This class defines the schema for the login table with column names for name, email, and password.

**TaskContract.java:**

* This class defines the schema for the tasks table with column names for task details like ID, name, priority, start date, end date, and description.

**TaskDao.java:**

1. addPerson(Task task):
   * This method inserts a new task into the "mytable" table.
   * It uses ContentValues to define the values to be inserted.
2. getAllTask():
   * This method retrieves all tasks from the "mytable" table.
   * It queries the table and returns an **ArrayList<Task>** containing task objects.

These classes and methods together manage the creation, insertion, updating, and retrieval of task and login data in the Goal Tracker app's SQLite database.

**SwipeToDeleteCallback.java:**

1. onMove(RecyclerView recyclerView, RecyclerView.ViewHolder viewHolder, RecyclerView.ViewHolder target):
   * This method is not used in this implementation.
   * It returns **false** to indicate that dragging is not supported.
2. onSwiped(RecyclerView.ViewHolder viewHolder, int direction):
   * This method is called when a swipe action is performed on a RecyclerView item.
   * It removes the swiped item from the adapter's data and updates the RecyclerView accordingly.

**Task.java:**

1. Constructors:
   * These constructors are used to create instances of the **Task** class with various combinations of parameters.
2. Getter and Setter Methods:
   * These methods are used to retrieve and modify the properties of a **Task** object, such as task name, priority, start date, end date, etc.

**TaskAdapter.java:**

1. onCreateViewHolder(ViewGroup parent, int viewType):
   * This method is called when a new RecyclerView item needs to be created.
   * It inflates the layout for the individual task item.
2. onBindViewHolder(TaskViewHolder holder, int position):
   * This method binds the data from the **taskList** to the UI elements of the ViewHolder.
   * It also sets up click listeners and handles launching the **SingleGoalActivity** when an item is clicked.
3. removeItem(int position):
   * This method displays an alert dialog to confirm the deletion of a task.
   * If confirmed, it removes the task from the list and updates the adapter and database accordingly.
4. getItemCount():
   * This method returns the number of items in the **taskList**.
5. TaskViewHolder:
   * This is an inner class that defines the ViewHolder for the task items.
   * It holds references to the UI elements of each task item, such as status icon, task name, and task date.

These classes and methods work together to enable swiping to delete functionality, bind data to the RecyclerView, and handle user interactions with the task items in the Goal Tracker app.

**Timeline**

The project adheres to a well-structured timeline:

| Task | 7th Week | 8th Week | 9th Week | 10th Week | 11th Week | 12th Week |
| --- | --- | --- | --- | --- | --- | --- |
| Design Phase |  |  |  |  |  |  |
| Development Phase |  |  |  |  |  |  |
| Implementation Phase |  |  |  |  |  |  |
| Testing and Submission |  |  |  |  |  |  |

**Snippents from the Application:**

Login Module

A screenshot of a phone

Description automatically generatedA screenshot of a phone

Description automatically generatedA screen shot of a phone

Description automatically generated

Home Screen

A screenshot of a cell phone

Description automatically generatedA screen shot of a phone

Description automatically generated

Goal handling

A screenshot of a phone

Description automatically generatedA screenshot of a phone

Description automatically generatedA screen shot of a phone

Description automatically generated

Contact us and Settings

A screenshot of a phone

Description automatically generatedA screen shot of a phone

Description automatically generatedA screen shot of a phone

Description automatically generated

**Conclusion**

The “Goal Tracking” App seamlessly bridges the gap in task management, offering a streamlined digital solution. Through well-defined classes and methods, the app delivers features that significantly enhance user productivity. By encompassing user-centric functionalities like goal setting, progress tracking, and data visualization, the app contributes to improved task organization and achievement. This project demonstrates a comprehensive grasp of Android development principles and showcases the ability to create functional and user-oriented applications.