IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Taskopia - REAL TIME EVENT TRACKING AND CONTROL

Vasudha Khandagale[1], Shivtej Pisal[2], Rehan Khan[3], Harsh Jagtap[4], Vishal More[5]

[1] Professor, Dept. of Computer Engineering, Sinhgad Academy of Engineering, Pune, Maharashtra

[2,3,4,5] UG Scholar, Dept. of Computer Engineering, Sinhgad Academy of Engineering, Pune, Maharashtra

***Abstract**:* In an increasingly digitized world where time management is paramount, Taskopia emerges as a robust solution designed to cater to the diverse needs of task organization and productivity enhancement. Developed using state-of-the-art technologies including Flutter, Dart, Riverpod, and Sqflite, Taskopia represents a fusion of innovation and practicality in the realm of mobile task management applications. Taskopia's primary objective is to provide users with a seamless and intuitive platform for managing tasks efficiently, regardless of their complexity or frequency. By harnessing the power of Sqflite, Taskopia ensures data persistence, enabling users to access and modify their task lists even in offline environments, thereby minimizing disruptions and maximizing productivity. Moreover, Taskopia integrates Firebase for OTP generation, enhancing the security and authentication protocols for user accounts. This integration not only fortifies the application against potential threats but also instills confidence in users regarding the confidentiality and integrity of their personal information. This paper delves into the intricate details of Taskopia's development journey, elucidating the architectural nuances, technological underpinnings, and key features that define its functionality. By comprehensively understanding Taskopia's design and implementation, users can leverage its capabilities to streamline their task management processes

***Index Terms* -** Event detection; cluster analysis; burst detection; Twitter; microblog analysis; social networks; data stream mining

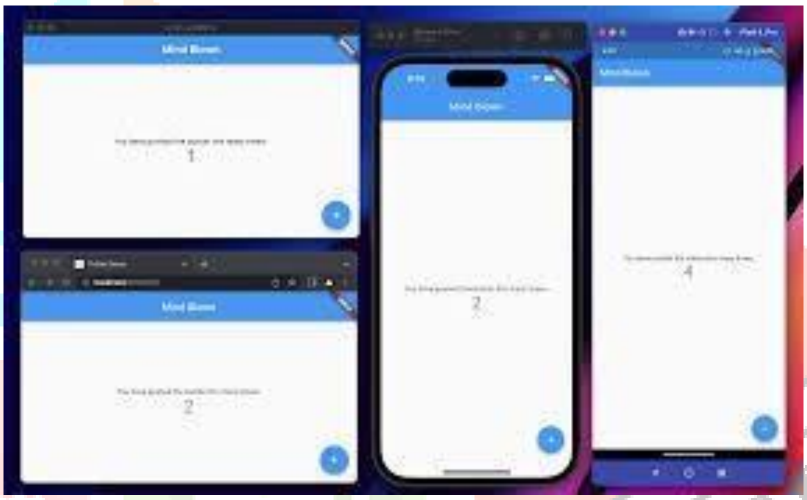
I. INTRODUCTION

In an era defined by relentless schedules and everincreasing demands on our time, effective task management has become more than a convenience— it's a necessity. Enter Taskopia, a cutting-edge mobile application meticulously crafted to address the modern challenges of task organization and productivity enhancement. Developed using a sophisticated blend of technologies including Flutter, Dart, Riverpod, and Sqllite, Taskopia represents a paradigm shift in the way individuals and organizations approach their daily tasks. Taskopia is more than just another task management app; it's a comprehensive solution designed to empower users to take control of their schedules, prioritize effectively, and achieve optimal productivity levels. Leveraging Sqflite's robust data persistence capabilities, Taskopia ensures that users can access their task lists seamlessly, whether they're online or offline, eliminating the frustration of being unable to manage tasks due to connectivity issues. Furthermore, Taskopia integrates Firebase for OTP (One-Time Password) generation, enhancing the security and reliability of user authentication. By implementing industry-leading security measures, Taskopia instills confidence in users, assuring them that their sensitive data is safeguarded against unauthorized access and breaches. This paper embarks on a journey to unravel the intricacies of Taskopia's development process, providing insights into its architecture, feature set, and technological foundations. By gaining a deeper understanding of Taskopia's inner workings, users can harness its full potential to revolutionize their task management workflows, boost efficiency, and ultimately, reclaim control over their time and productivity.

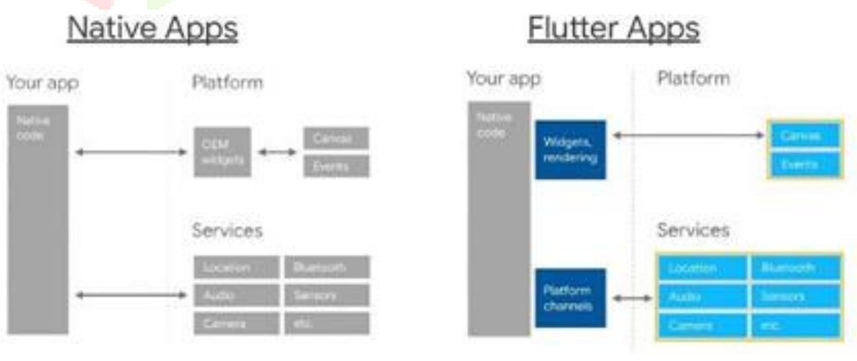
II. LITERATURE REVIEW

Flutter is a UI toolkit and SDK which you can use to build applications. Flutter is open source and you can use it to build highly performant mobile and desktop apps. Flutter is Cross-Platform: Software is cross-platform when it is available for different Operating Systems. You want your product to have such cross-platform ability so users on any device can use your product comfortably. Having support for desktop, mobile, and web platforms is tough. For desktop, you will need to write code for macOS (with Swift), Linux (with C), and Windows (with C++). For mobile, you will need to write code for Android (with Kotlin/Java and XML) and iOS (with Swift). To make your product accessible as a website, you have to use HTML, CSS, and JavaScript. Or, use any frontend JavaScript framework like Angular, React, or Vue. Making applications for the various desktop and mobile operating systems requires separate SDKs and skillsets. In the past, you would've needed to hire developers who were proficient at each platform to implement your app on each of those platforms. This is expensive. To add a feature, you would've needed to update the code across all these platforms which is tedious.

How Flutter is cross-platform: Flutter code can run on desktop, mobile, and web platforms. So, you don't need to hire developers for each platform. You need to write the code only once in Flutter and you can rest assured that the app will work across the other platforms. So, Flutter is cheap. Adding features to your app is fast because you only have to make code updates once in Flutter and that's all.



The Flutter framework gives you APIs for painting andevents (through widgets). It also provides APIs for platform-specific services (through method channels).So Flutter *exposes everything* to you to build the app in any way you want it. That's how cross-platform works.



2.1 Flutter Has Great State Management:

State refers to data that your application needs torender its UI at any point in time. It could be user- generated or from your servers or backend. In general, Flutter has two types of widgets: Stateless Widgets and Stateful Widgets. Anytime you will need to update the UI based on state, simply use aStateful Widget ahead of time. Then to update the UI, call setState() anywhere inside the State class of aStateful Widget and Flutter will rebuild the UI tree This setState() mechanism turns out to be the most efficient out there. It is also the React style of updatingUI based on state. UI programming should be declarative and setState() simply advocates that.

You should call setState() after asynchronous code has been executed (if any). This is because it is part of the main UI thread and blocking calls can hijack the UI rendering process. If you have processes that are highly CPU intensive in a Flutter app, consider using Dart Isolates to carry out those processes, then call setState() with the data you get when processing is complete.

Sometimes, you may need access to state not scoped to a particular widget. At that point, you need a State Management architecture. There are a good number of them to choose from. These architectures provide state at an app-wide level without reconfiguring in each widget. Available options include Provider, Riverpod, Redux, Inherited Widget, Stacked, and many more. Another advantage of these architectures is that a good number of them permit good separation of concerns and dependency injection (or inversion of control).In simpler terms, separation of concerns gives you the ability to write UI-specific code separate from logic-specific code. This way, if you want to debug, or change the packages or APIs you're using, you'll do them easily from one place (without fear of damaging the codebase). Besides, it promotes clean code too. Dependency Injection involves the use of services or something similar to obtain what widgets need to work without the widgets configuring the services themselves. Some architectures like Stacked permit you to manage app-wide states without the Build Context. This pattern is useful because, in Stateless Widgets, the BuildContext is available only within the build method and you might need to use stuff outside it. So you could do other things like BottomSheet, Navigation, Toast, and so on without Build Context.

2.2 Flutter Provides a Great Developer Experience:

There are many reasons the developer experience with Flutter is awesome. Here are a few of them:Flutter has only one programming language – Dart. Programming on its own is a demanding activity. Frameworks, libraries, and tools shouldn't make coding tougher. When coding for some platforms you usually write code in more than one programming language. For example, while coding for frontend web, you will alternate between HTML, CSS, and JavaScript files to manage a given portion of your webpage. Flutter makes coding easy because you write in just one programming language: Dart. So most of the time, the logic and UI properties for a given portion of your Flutter app will be in the same Dart file. Also, Flutter and Dart are written in plain English. Widget names and their properties reflect what they are. While coding Flutter, you are less likely to have any headaches understanding a given widget and or its use case(s). Flutter has hot-reloading. Dart is a modern programming language that comes with an Ahead-Of-Time (AOT) and a Just-In-Time (JIT) compiler.

The JIT compiler gives the feel that Dart is rather interpreted in runtime. In other words, it makes it possible for changes to Dart files to reflect immediately without needing to recompile the file, as is the case with some programming languages like C or Java.

Flutter leverages the JIT for development. Flutter ships a Dart VM to the platform in which the app's code is hosted. That way, when you launch the flutter run command, pressing r in that terminal ships in the changes in dart files, without recompiling the whole app. This instantaneous, yet *cross-platform* change-reflecting feature of Flutter is called hot-reloading.

Flutter also has hot-restarting. Hot restarting is achieved by pressing R (uppercase instead of lowercase this time). *Hot restarting* restarts the app entirely. You need to do a hot-restart instead if you change some important parts of the Flutter code. For example, changing class declarations or their super classes will need a hot restart.

III. METHODOLOGY

3.1 Project Scope Definition:

- **Identify the requirements:** The project entails the development of a Flutter-based mobile application named Taskopia, aimed at facilitating efficient task management for users across various domains.
- **Define scope:** Taskopia will offer a comprehensive suite of features to enable users to organize, prioritize, and track their tasks seamlessly. The application will leverage Sqllite for data persistence, ensuringthat users can access their task lists even in offline mode. Additionally, Taskopia will integrate Firebase for OTP generation, enhancing security and authentication for user accounts. The primary focus of Taskopia is to provide an intuitive and user-friendly interface for managing tasks effectively, thereby improving productivityand goal attainment.

3.2 Data Collection:

- **Collection of task-related information:** Gather a dataset of task-related data, including task names, descriptions, deadlines and priority levels. Thisinformation will be used to train the taskmanagement algorithms and optimize taskorganization.
- **User interaction data:** Capture user interactions within the Taskoopia application, such as task creation, completion, and modification. This data will provide insights into user behavior and preferences, allowing for personalized task recommendations and optimizations.
- **Firebase authentication data:** Collect authentication data generated by Firebase for user account management and security purposes. This data will include OTP (One- Time Password) logs and user authentication events, enabling analysis of user authentication patterns and detection of potential security threats.
- **Sqllite for local data storage:** Utilize Sqfliteto store task-related information locally on the device. This will enable Taskoopia toprovide offline access to task lists and ensuredata persistence even when the device is not connected to the internet. Sqflite's lightweight and efficient SQLite database implementation make it an ideal choice for storing and retrieving task data on mobile devices.

.

3.3 Tools Used:

- **Android Studio:** For developing the Android application.
- **Flutter SDK and Dart:** Utilized for developing the Taskopia application, providing a robust framework for cross- platform mobile app development.
- **Sqllite:** Employed for local data storage within the Taskopia app, ensuring seamless access to task-related information even in offline mode.
- **Firebase:** Integrated for various functionalities including OTP generation, user authentication, and secure storage ofuser data.
- **Riverpod:** Utilized for state management within the Taskopia application, enabling efficient management of app-wide data and UI states.
- **Git:** Version control system for collaborativedevelopment.

3.4 EXPERIMENTAL DESIGN:

- **Task Management Algorithm Training:** Train machine learning models for task management algorithms using datasets comprising task-related information. Experiment with different algorithms such as decision trees, random forests, or neural networks to optimize task organization and prioritization.
- **Testing:** Conduct extensive testing of the Taskopia application to evaluate its functionality, performance, and user experience. Test under various scenarios andconditions to identify and rectify any potential issues or bugs.
- **Offline Mode Testing:** Evaluate the performance of Taskopia's offline mode functionality, ensuring

seamless access to task lists and data persistence even without an internet connection.

- **Firebase Integration Testing:** Test the integration of Firebase services, includingOTP generation and user authentication, to verify their functionality and reliability underdifferent usage scenarios.
- **Riverpod State Management Testing:** Validate the effectiveness of Riverpod forstate management within the Taskoopiaapplication, ensuring consistent and efficientmanagement of app-wide data and UI states.

3.5 RELEVANT PROCEDURES:

- **Task Management Functionality:** Preprocess task-related data: Standardize task names, descriptions, and deadlines for uniformity. Implement algorithms to prioritize tasks based on user preferences anddeadlines.
- **Local Data Storage:** Utilize Sqllite toestablish a local database within the Taskopia application. Implement CRUD (Create, Read, Update, Delete) operations to manage task data efficiently.
- **Firebase Integration:** Integrate Firebase services for user authentication, ensuring secure access to Taskopia's features. Implement Firebase OTP generation for user account verification and authentication.
- **Riverpod State Management:** Utilize Riverpod for state management within the Taskopia application, facilitating seamless communication between different components and widgets.

IV. IMPLEMENTATION

4.1 Setting up the Development Environment:

Setting up the development environment involves installing Flutter SDK along with Dart, the programming language used forFlutter development. Flutter provides tools for building, testing, and debugging cross- platform mobile applications. Once installed,a new Flutter project is created, configuring permissions for local data storage, Firebase integration, and networking as required by the application.

4.2 Integrating Task Management Functionality:

Task management functionality can be implemented using Flutter packages compatible with Sqllite for local data storageand Firebase for backend services. These packages offer APIs for CRUD operations on local databases and interacting with Firebase services. Integration involves adding the package dependencies to the project, configuring settings, and implementing codeto manage tasks, prioritize them, and ensure synchronization between local and remotedata sources.

4.3 Implementing Firebase Integration:

Utilizing Firebase services allows the application to leverage features such as authentication, data storage, and cloud functions. A Firebase project is set up and configured with the necessary services enabled, including Firestore for storing task-related data securely. Integration involves adding the Firebase SDK to the project, configuring authentication methods such as OTP generation, and implementing code to authenticate users, store and retrieve task data from Firestore, and handle user sessions.

4.3 User Interface Development:

Designing and implementing a user-friendly interface iscrucial for the application's usability. The UI should allow users to view, create, update, and delete tasks seamlessly. The interfaceshould be intuitive and responsive, followingMaterial Design guidelines for consistency across different devices and screen sizes. Widgets such as List Tile, List View, and Floating Action Button can be utilized tocreate a cohesive and visually appealing userexperience. Dynamic colors functionality canbe integrated to allow users to customize theapp's color scheme according to their preferences.

4.4 Testing and Refinement:

Testing the application extensively on different devices ensures compatibility and reliability across a wide range of hardware configurations. Unit tests, integration tests, and widget tests are conducted to verify the functionality of individual components and the overall application. Real-world testing with beta users helps validate the effectiveness of task management features and gather feedback for refinement. Iterative updates and bug fixes are implemented based on user feedback andtesting results to improve the application's functionality and user experience.

V. ARCHITECTURE

The architecture of the Taskoopia task management application comprises several key components designed to facilitate seamless task organization and productivity enhancement for users across various domains. These components include the Task Manager, User Interface (UI), Data Management, and Firebase Integration.

5.1 Task Manager:

The Task Manager component forms the core of the application, responsible for managing tasks, prioritizing them, and ensuring synchronizationbetween local and remote data sources. It encompasses functionalities such as task creation,modification, deletion, and completion, as well astask prioritization based on user preferences and deadlines.

5.2 User Interface:

The UI component encompasses various screens and elements designed to provide a user-friendly and intuitive experience for Taskoopia users. It includes screens for task lists, task details, task creation, settings, and user profile management. The UI design focuses on clarity, simplicity, and responsiveness, with minimalist design elements and intuitive navigation to enhance user engagement and productivity.

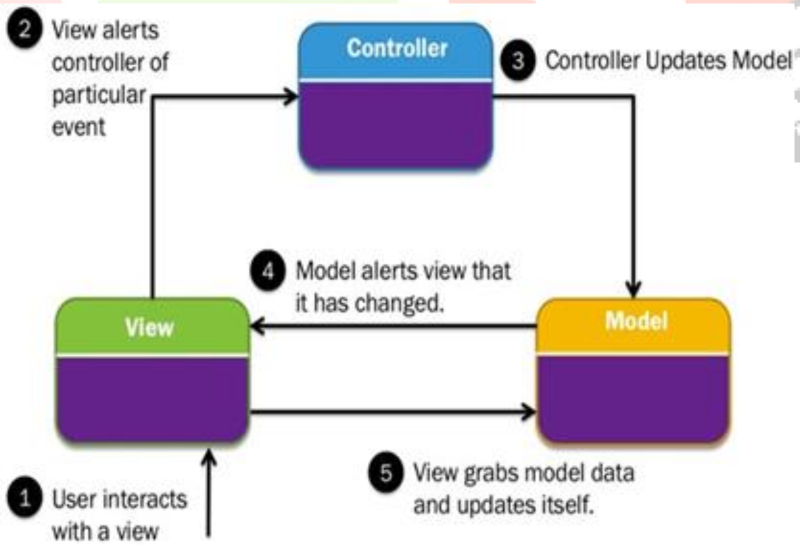

2 View alerts controller of particular event
Controller
3 Controller Updates Model
4 Model alerts view that it has changed.
View
Model
1 User interacts with a view
5 View grabs model data and updates itself.

Fig: MVC Architecture

5.3 Data Management:

The Data Management component handles the storage and retrieval of task-related data within the Taskoopiaapplication. It utilizes Sqflite for local data storage, ensuring seamless access to task lists even in offline mode. Additionally, Firebase integration is employed for data synchronization and real-time updates, enabling users to access their tasks across multiple devices and platforms.

5.4 Firebase Integration:

Firebase integration encompasses various services offered by Firebase, including authentication, cloud Firestore, and cloud functions. Authentication services are utilized for user account management and security purposes, ensuring secure access to Taskopia's features. Firestore serves as the backend database for storing and retrieving task-related data securely, whilecloud functions enable serverless computation for taskssuch as data validation and processing.

Overall, the architecture of Taskopia is designed to provide a robust, scalable, and user-centric solution fortask management, empowering users to organize theirtasks effectively and achieve their goals with ease.

5.5 MATHEMATICAL MODEL:

Let S represent the entire system, where

$$S=I,P,O.$$

Given:

- I (Input): The input to the system consists of task-related information such as task names, descriptions, deadlines, and user preferences.
- P (Procedure): The procedure involves prioritizing tasks based on their deadlines anduser-defined preferences.
- O (Output): The output of the system is the organized task list, with tasks arranged inorder of priority.
- T: Set of tasks
- D: Set of deadlines
- Pi: Priority level assigned to task �i

The Mathematical Model:

$$Priority=f(T,D,Pi)$$

Where: $Priority=1+e-(\sigma di-\mu)1$

$$\max-di=D\max-TDi-T$$

$$\mu=n\sum i=1nPi$$

$$\sigma=n\sum i=1n(Pi-\mu)2$$

5.6 CHALLENGES ENCOUNTERED AND ADDRESSED:

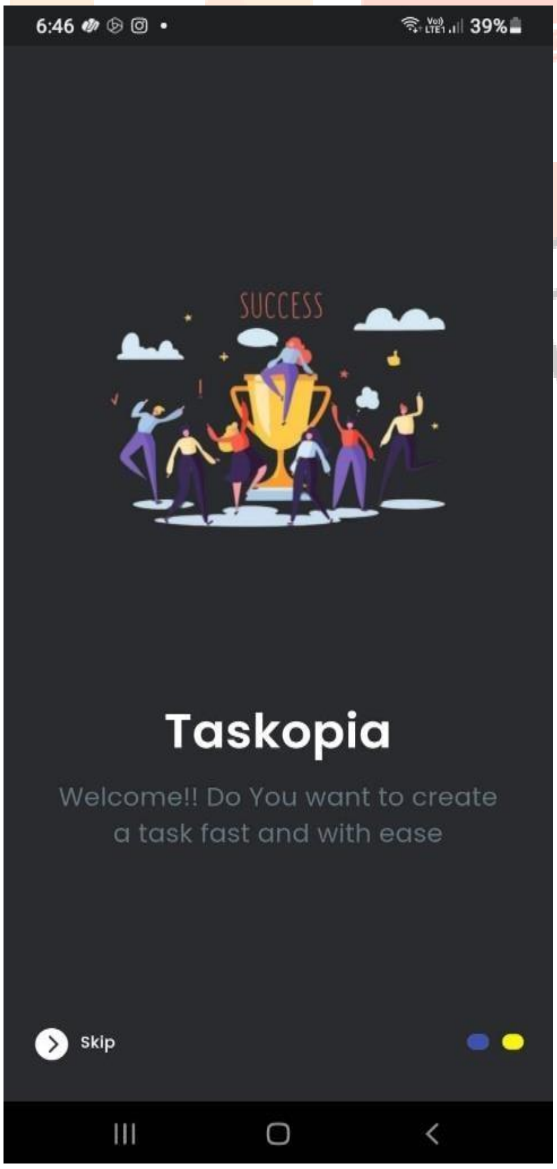
- **Task Organization and Prioritization:**
  Managing tasks effectively in a dynamic environment posed challenges, especially when dealing with a large number of tasks with varying priorities and deadlines. This was addressed by implementing efficient algorithms for task organization and prioritization, considering factors such as deadlines, importance, and user preferences.
- **Offline Functionality:**
  Ensuring seamless access to task lists and data persistence in offline mode presented challenges, particularly in scenarios with limited or no internet connectivity. This was addressed by utilizing Sqflite for local data storage and synchronization mechanisms to ensure that users can access their tasks even without an internet connection.
- **User Authentication and Security:**
  Implementing robust authenticationmechanisms and ensuring data security wereparamount to protect user privacy and prevent unauthorized access to sensitive information. This was addressed by integrating Firebase Authentication for secure user authentication and implementing encryption techniques to safeguard user databoth in transit and at rest.
- **Learning Curve:**
  Flutter requires developersto learn Dart programming language and theFlutter framework, which may have a steeperlearning curve for those unfamiliar with them.
- **Versioning and Updates**:
  Keeping up with Flutter updates and managing version compatibility with packages and dependencies can sometimes
  lead to challenges, especially for larger projects withmultiple contributors.

VI. RESULTS

6.1 OnBoarding Page:

The OnBoarding widget is the central componentof the Taskoopia app, providing users with a streamlined dashboard to manage tasks efficiently.Here's a brief overview:

- **AppBar and Search Bar:**
  The app bar displays the title "Dashboard" and an "Add" button for adding new tasks. Below it, a search bar allows users to find specific tasks quickly.
- **Task Tabs:**
  Two tabs, "Pending" and "Completed," enable users to switch betweenpending and completed tasks easily. The selected tab is highlighted for clarity.
- **Task Lists:**
  The tab bar view displays tasks corresponding to the selected tab. "Pending" shows tasks to be completed, while "Completed" lists tasks already finished.
- **Tomorrow and Day After Tomorrow Lists:**
  Sections at the bottom provide quick previewsof tasks scheduled for the next two days.
- **UI Elements:**
  The UI features a clean, modern design with minimalist elements for auser-friendly experience. Icons from Flutter Vector Icons guide users, and styled text fieldsand buttons maintain consistency.
- **Functionality:**
  Users can perform various actions such as viewing, adding, editing, and deleting tasks seamlessly. Tabs and scrollable lists facilitate easy navigation between task sections.



Taskopia

Welcome!! Do You want to create a task fast and with ease

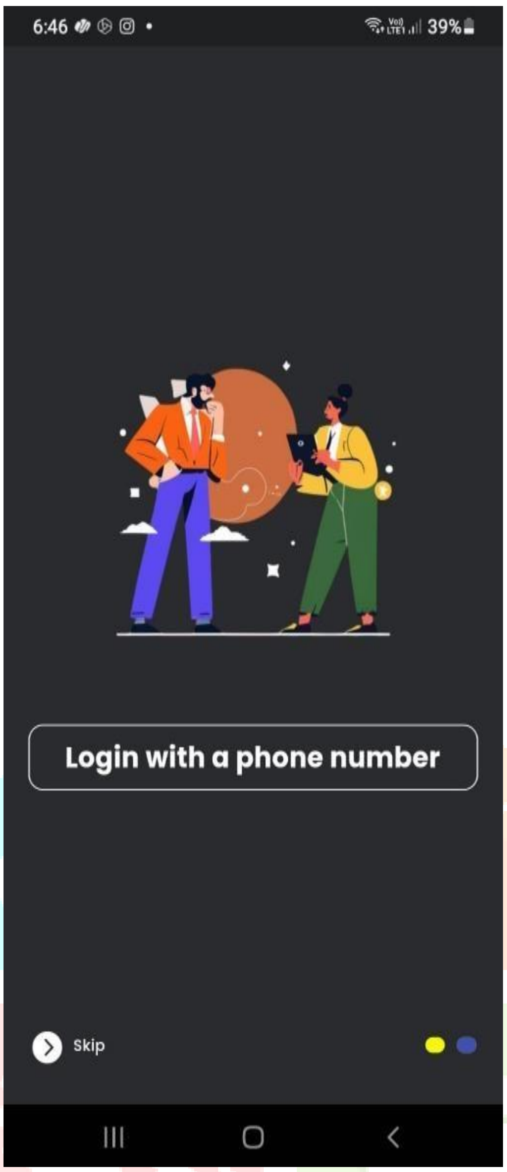
Overall, the OnBoarding offers a compact yet effective interface for efficient task management in Taskopia, ensuring users stay organized and productive.

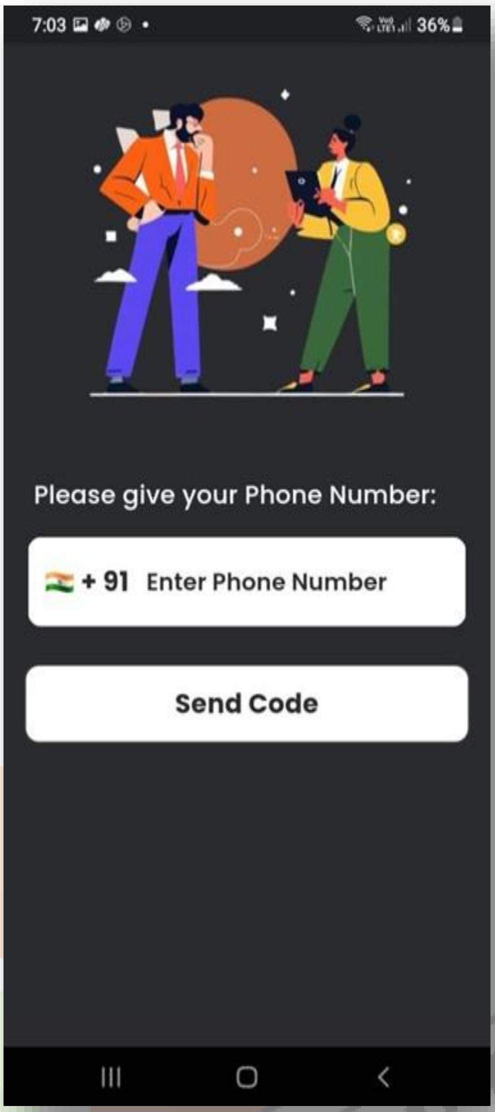
6.2 Login Page:

The Login Page widget serves as the entry point for users to authenticate and access the Taskopia app.

- **User Authentication:** The LoginPage allowsusers to input their phone number forauthentication purposes.
- **Country Selection:** Users can select theircountry using a dropdown menu, which displays the flag emoji and phone code of theselected country
- **UI Elements:** The UI features an image at thetop, providing visual appeal. Below it, a text prompt asks users to input their phone number. The phone number input field is accompanied by a country selection dropdown and styled accordingly for consistency.
- **Send Code Button:** A button labeled "Send Code" enables users to proceed with authentication. Upon tapping this button, users are navigated to the OTP verification page (OtpPage).
- **Scrollable Content**: The content of the LoginPage is contained within a scrollable view, ensuring compatibility with devices of various screen sizes.
- **Validating Logic**:
  The Otp Page includes validation logic to ensure that users enter a complete 6-digit OTP code before proceeding. Upon entering the complete OTPcode, users can proceed with verification.

- **Scrollable Content:**
  Similar to the Login Page, the content of the Otp Page iscontained within a scrollable view, ensuring compatibility with devices of various screen sizes.

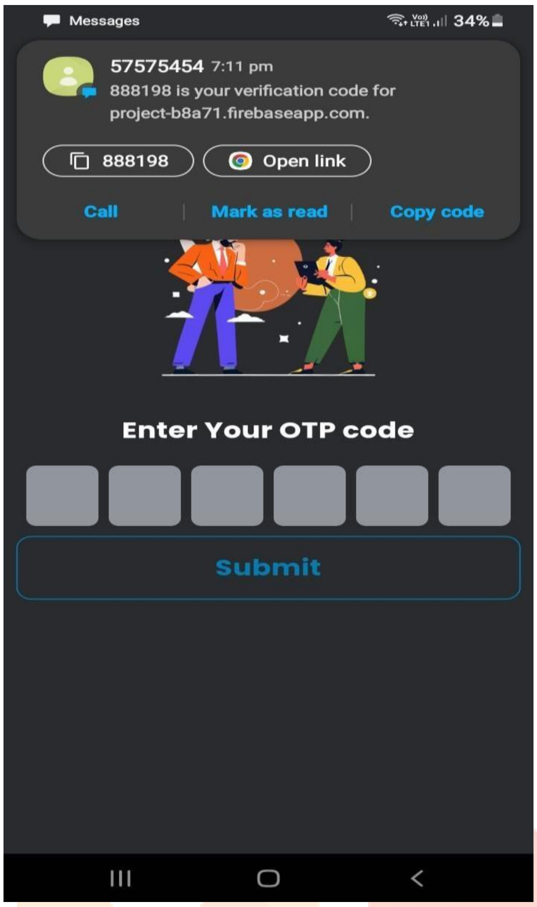




6.3 OTP PAGE:

The Otp Page widget is responsible for facilitating OTP (One-Time Password) verification during the login process in the Taskopia app. Here's a concise summary of its features and functionality:

- **OTP Input Field:**
  The main component of the OtpPage is an OTP input field where users can enter their OTP code. The input field is designed to accept a 6-digit OTP code.
- **UI Elements:**
  The UI features an image at the top, providing visual appeal and branding consistency. Below it, a text prompt instructs users to enter their OTP code.
- **OTP Input Field:**
  The OTP input field is implemented using the Pinput package, allowing users to input their OTP code securely. The field automatically moves the cursor to next digit after each input, providing a seamless user experience.

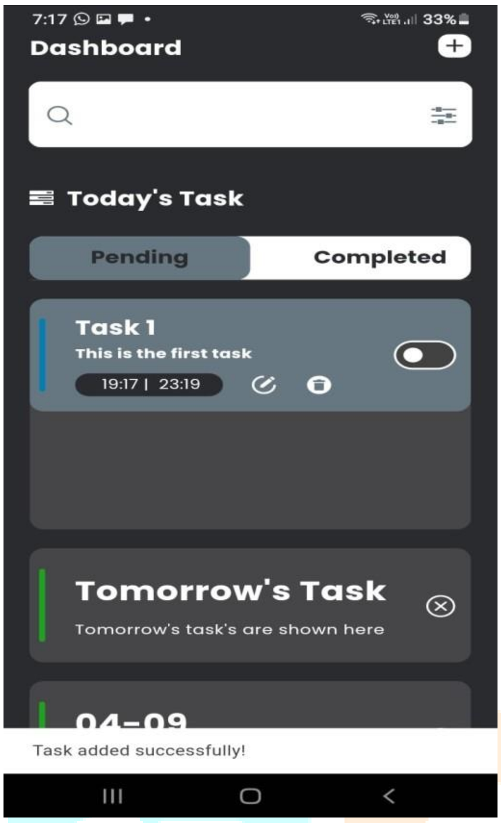


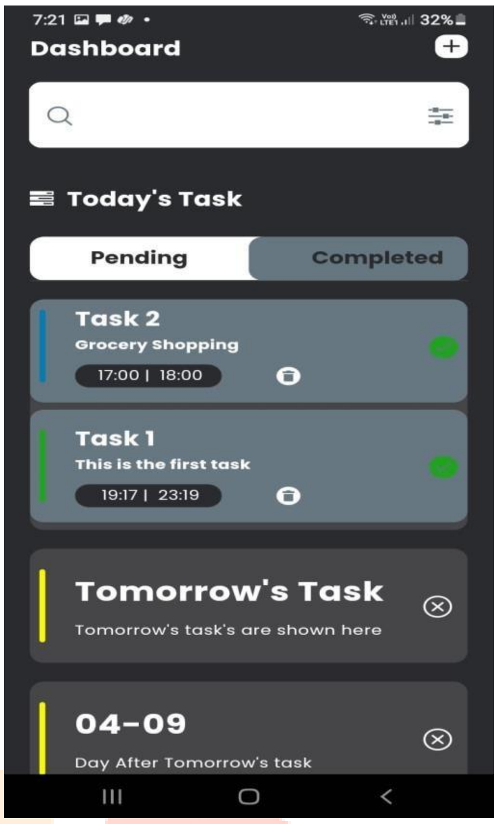
Overall, the Otp Page offers a straightforward and user- friendly interface for OTP verification, facilitating a secure login process for users in Taskopia.

6.4 Home Page:

The Home Page widget serves as the central hub for task management in the Taskopia app. Here's a concise overview of its features and functionality:

- **AppBar:**
  The app bar displays the title "Dashboard" and an "Add" button for creating new tasks. Tapping the add button navigates users to the task creation page.
- **Search Bar:**
  Below the app bar, there's a search bar allowing users to search for specific tasks. It enhances task organization and retrieval.
- **Task Tabs:**
  Two tabs, "Pending" and "Completed," enable users to switch between pending and completed tasks effortlessly. Each tab displays the respective tasks.
- **Task Lists:**
  The tab bar view presents task lists corresponding to the selected tab. "Pending" displays tasks yet to be completed, while "Completed" lists tasks already finished.
- **UI Elements:**
  The UI boasts a clean and modern design with minimalist elements, ensuring an intuitive user experience. Icons from the Flutter Vector Icons package provide visual cues for actions.
- **Functionality:**
  Users can view, add, edit, and delete tasks seamlessly. The HomePage facilitates smooth navigation between different task sections, enhancing productivity.


7:17 33%
Dashboard
Today's Task
Pending Completed
Task 1
This is the first task
19:17 | 23:19
Tomorrow's Task
Tomorrow's task's are shown here
04-09
Task added successfully!

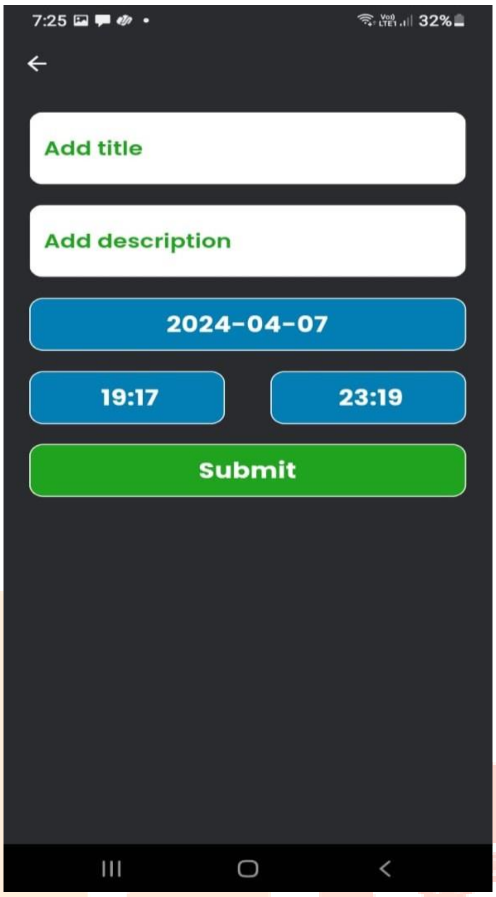

7:21 32%
Dashboard
Today's Task
Pending Completed
Task 2
Grocery Shopping
17:00 | 18:00
Task 1
This is the first task
19:17 | 23:19
Tomorrow's Task
Tomorrow's task's are shown here
04-09
Day After Tomorrow's task

6.5 Add Task Page:

The Add Task page allows users to create new tasks with various details such as title, description, scheduled date, start time,
and finish time. Here's a brief overview of its features and functionality:

- **App Bar:**
  The app bar is transparent with no elevation, providing a clean interface for taskcreation.
- **Text Fields:**
  Users can input the title and description of the task using custom text fields.
- **Date and Time Pickers:**
  Custom outline buttons allow users to set the scheduled date,start time, and finish time of the task. Date andtime pickers facilitate easy selection ofspecific dates and times.
- **Submit Button:**
  Upon filling out all requiredfields, users can submit the task details by tapping the "Submit" button. The task is thenadded to the task list.
- **Validation:**
  The page validates that allnecessary fields are filled out before allowingsubmission. If any field is empty, the submission fails, and an error message is printed.
- **State Management:**
  The page utilizes Riverpod for state management, ensuring efficient updates and changes to task-related data.

Overall, the Add Task page streamlines the process of creating new tasks, providing users with a convenient and intuitive interface within the Taskopia app.



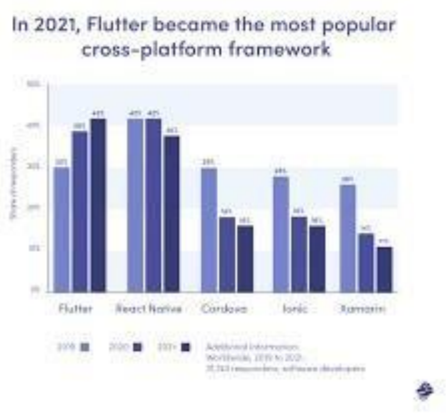
VII. DISCUSSION

Analyzing the outcomes of implementing Taskopia- a real-time event tracking and control application -in the context of our project objectives sheds light on the superiority of Flutter over traditional languages like Kotlin, Swift, or React Native. Our primary objectives, as outlined in the project introduction, focused on optimizing task management processes, enhancing performance, and delivering a seamless experience for users.

First and foremost, Flutter's cross-platform support proved instrumental in streamlining task management app development effectively. Leveraging Flutter's single codebase approach, we could create a unified user interface for both Android and iOS platforms, eliminating the need for separate codebases. This unified development approach significantly reduced development time and effort, enabling us to provide a consistent user experience across different devices.

Secondly, Flutter's performance and efficiencyallowed us to enhance the responsiveness and reliability of our task management appWith Flutter's fast rendering engine and native performance, we could implement complex UI interactions seamlessly, ensuring a smooth and responsive user experience. This native performance capability surpassed other cross-platform frameworks, resulting in enhanced app performance and user satisfaction.

Furthermore, Flutter's extensive widget library and customizable UI components empowered us to deliver convenient functionalities for Taskoopia users. By utilizing Flutter's built-in widgets, we could design intuitive and visually appealing interfaces for managing tasks, setting reminders, andorganizing lists. Flutter's hot reload feature accelerated the development process, enabling us to iterate quickly and incorporate user feedback effectively, thereby improving overall user satisfaction and engagement.



The implications of these findings underscore Flutter's superiority in mobile app development, particularly for task management solutions like Taskopia. Flutter's versatility, performance, and productivity enhancements offer significant advantages over traditional languages like Kotlin, Swift, or React Native. By adopting Flutter for our project, we not only achieved our objectives efficiently but also positioned ourselves at the forefront of mobile app development innovation.

However, it's essential to acknowledge potential challenges associated with Flutter development, such as plugin compatibility issues or a learning curve for developers transitioning from other frameworks. Despite these challenges, the overall benefits and advantages offered by Flutter far outweigh any potential drawbacks, making it the preferred choice for modern mobile app development projects.

By comparing our project results with existing literature and research on mobile app development frameworks, we can affirm Flutter's superiority andits potential to revolutionize the way mobile apps are built and deployed. As Flutter continues to evolve and gain traction in the developer community, its impact on the mobile app development landscape ispoised to grow, solidifying its position as the leadingchoice for cross-platform development.

VIII. DRAWBACKS

8.1 Larger App Size:

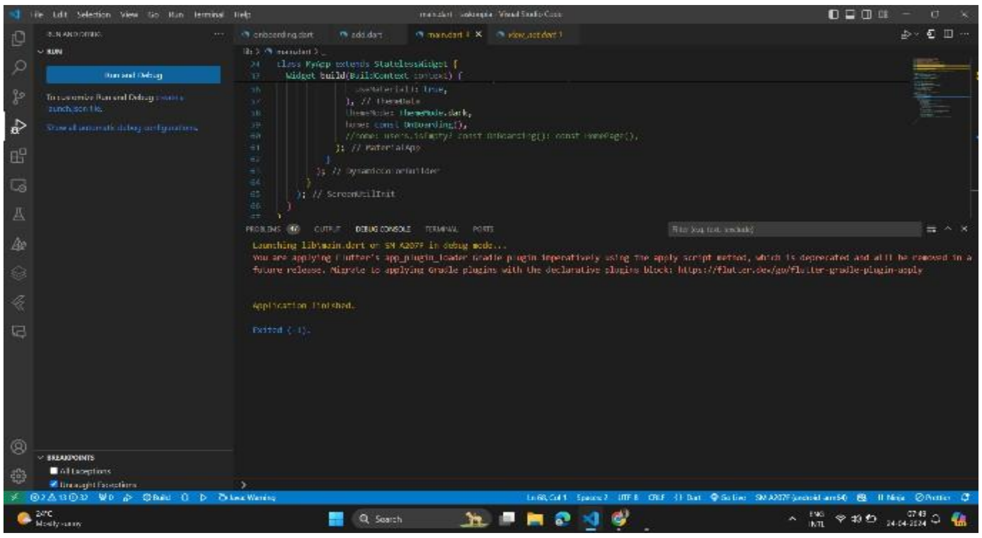
One of the notable downsides of Flutter is the size of the resulting apps. Flutter apps tend to be larger compared to their native counterparts. Thiscan be a concern for users with limited storage space on their devices or for apps that need to be downloaded over a mobile network. The inclusion of the Flutterframework contributes to this larger size.

8.2 Learning Curve:

Developers who are not familiar with Dart, the programming language used by Flutter, may face a learning curve. For those already proficientin Java/Kotlin for Android or Swift/Objective-C for iOS, transitioning toDart can be challenging and time- consuming.

8.3 Platform Updates:

Both Android and iOS receive regular updates, and Flutter must adapt to these changes. Sometimes, there may be a delay in Flutter’s support for newplatform features, potentially causing issues for app maintenance.

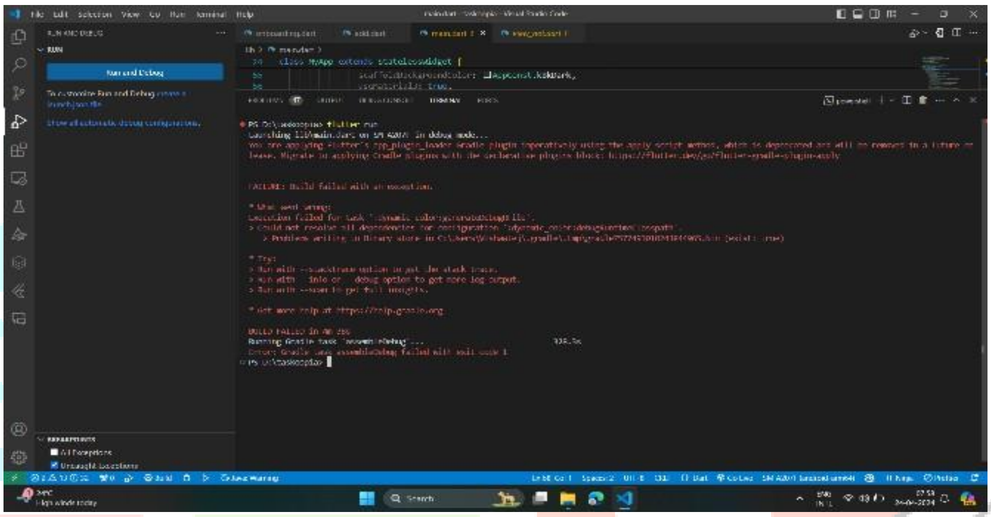


```
class MyApp extends StatelessWidget {
  Widget build(BuildContext context) {
              useMaterial3: true,
            ), // ThemeData
            themeMode: ThemeMode.dark,
            home: const OnBoarding(),
            //home: users.isEmpty? const OnBoarding(): const HomePage(),
          ); // MaterialApp
        }
      ); // DynamicColorBuilder
    }
  ); // ScreenUtilInit
}
```

```
Launching lib\main.dart on SM A207F in debug mode...
You are applying Flutter's app_plugin_loader Gradle plugin imperatively using the apply script method, which is deprecated and will be removed in a
future release. Migrate to applying Gradle plugins with the declarative plugins block: https://flutter.dev/go/flutter-gradle-plugin-apply

Application finished.

Exited (-1).
```



```
class MyApp extends StatelessWidget {
              scaffoldBackgroundColor: AppConst.kBkDark,
              useMaterial3: true,
```

```
PS D:\taskopia> flutter run
Launching lib\main.dart on SM A207F in debug mode...
You are applying Flutter's app_plugin_loader Gradle plugin imperatively using the apply script method, which is deprecated and will be removed in a future re
lease. Migrate to applying Gradle plugins with the declarative plugins block: https://flutter.dev/go/flutter-gradle-plugin-apply

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':dynamic_color:generateDebugRFile'.
> Could not resolve all dependencies for configuration ':dynamic_color:debugRuntimeClasspath'.
   > Problems writing to Binary store in [illegible] (exists: true)

* Try:
> Run with --stacktrace option to get the stack trace.
> Run with --info or --debug option to get more log output.
> Run with --scan to get full insights.

* Get more help at https://help.gradle.org

BUILD FAILED in [illegible]
Running Gradle task 'assembleDebug'...                    [illegible]
Error: Gradle task assembleDebug failed with exit code 1
PS D:\taskopia>
```

IX. CONCLUSION

Taskopia leverages modern technologies such as Flutter, Dart, Riverpod, SqLlite, and Firebase to streamline task management workflows and enhance user productivity. The use of Flutter allows for the creation of a single codebase that can run on multiple platforms, ensuring consistent performance and user experience across different devices. Additionally, Dart provides a robust and efficient programming language, while Riverpod facilitates state management, SqLlite enables local database storage, and Firebase offersreal-time data synchronization and authenticationservices.

By utilizing these technologies, Taskopia offers features such as task creation, organization, and tracking in real-time, providing users with a comprehensive solution for managing their tasks efficiently. The implementation of Taskopia demonstrates the potential of technology-driven solutions to address traditional task management challenges and improve productivity in various contexts.

Moving forward, future iterations of Taskopia may focus on refining existing features, incorporating user feedback, and expanding functionality to meet the evolving needs of users. Additionally, efforts may be directed towards optimizing performance, ensuring data security and privacy, and preparing for potential deployment and publication.

X. REFERENCES

1) AI based Event Management Web Application: https://ieeexplore.ieee.org/document/9850551/metrics#metrics
2) TaskCO: Android App for Task Management.https://ieeexplore.ieee.org/document/10039511
3) Real-time Student Management Application UsingGoogle Firebase and Android Studio.https://ieeexplore.ieee.org/document/9498494
4) Hikester - The Event Management Application.https://ieeexplore.ieee.org/document/8418114
5) An Android Application System to Provide Real-time Notification to End Users Using Firebase.https://ieeexplore.ieee.org/document/10134323