# TASK MANAGER MOBILE APPLICATION AND ENSURING DEVICE SECURITY

***Mahkamov Anvarjon Abdujabborovich***
*International Islamic Academy of Uzbekistan*
*"Modern information and communication*
*Technologies" department, associate professor, PhD*
*mahkamovanvar2020@gmail.com*

**Defining project requirements**: The first and most important step in the application development process is to define project requirements. This step serves as a solid foundation that ensures the successful operation of the application. Initially, the functional and non-functional requirements of the system were identified.

Functional requirements define user actions and the tasks that the system must perform. Functional requirements for the Kyousuke application include: user login and registration; create, edit, and delete personal and team tasks; assign tasks to team members; track status changes in real time; prove task completion through a Proof of Work system; receive notifications, and view statistics[4].

Non-functional requirements include parameters that affect the quality of system performance, security, stability, and user interface efficiency. These requirements ensure the application's fast performance, real-time synchronization, fault tolerance, and user experience. In particular, non-functional requirements include the ability to instantly display data on the screen through Firebase Firestore StreamProvider, ensuring data security through Firestore Security Rules, and creating an additional layer of protection through PIN code and biometric authentication.

The requirements definition phase of the project involved a detailed analysis of user needs. This involved studying the organization's team, their technical

capabilities, and the functionality they expected from the app. At the same time, the project's constraints - device resources, Android platform requirements, and security standards - were also identified[6].

**Architecture and design phase**. After the requirements definition phase, the technical architecture and design of the application were developed. At this stage, the Clean Architecture principle was chosen, dividing the system into three layers: the Data Layer, the Domain Layer, and the Presentation Layer[6]. Such an architecture ensures that each layer operates independently and that new features will not affect other layers when added in the future[1-3].

The database structure, UI elements, and the connections between Firebase and Flutter are also defined in this phase. Firestore collections - tasks, teams, users, notifications - and their fields are designed in this phase. The Riverpod library was chosen for state management because it allows streaming Firestore data to the UI in real time via a StreamProvider[4-7].

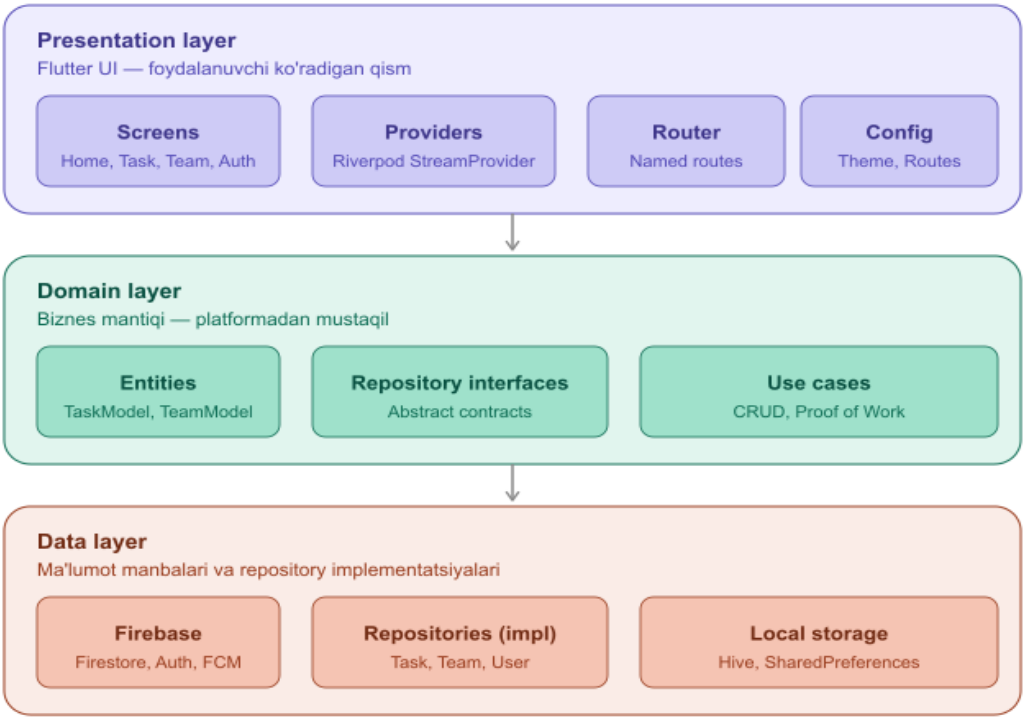


Figure 1. Program architecture

The design phase adopted Material Design 3 principles for user-friendliness and a modern look [12] The main color palette was purple (#8E7FFF), a dual theme (Light/Dark), and the Poppins font family were chosen. The navigation system, screen transitions, and animations were also planned at this stage.

Coding and Integration. After the architecture and design phase was completed, the coding phase began. In this phase, each module was developed separately and then their interoperability was ensured. Quality control and optimization were continuously carried out during the coding process[7].

After the code for the software modules was written, the integration phase began. At this stage, all modules and components were combined and the system was fully operational. In particular, Firebase Authentication via AuthProvider, Firestore task collection via TaskProvider, teams collection via TeamProvider, and FCM push notification system via NotificationService were integrated with each other.

Work was also done to document the code and align it to standards. Code quality control was provided through Dart's strong typing and analysis options.yaml. Clean and organized code makes it easier to expand the system, fix bugs, and add new features in the future.

Database integration. Seamless integration between the application interface and the Firebase Cloud Firestore database ensures system efficiency. The Cloud Firestore NoSQL database is used to store user data, tasks, teams, and notifications.

Every user action - creating a task, changing status, adding a team member, submitting a report - is transmitted to the database in real time. At the same time, changes in the database are immediately reflected in the interfaces via StreamProvider. This process improves the user experience and optimizes system performance[9-11].

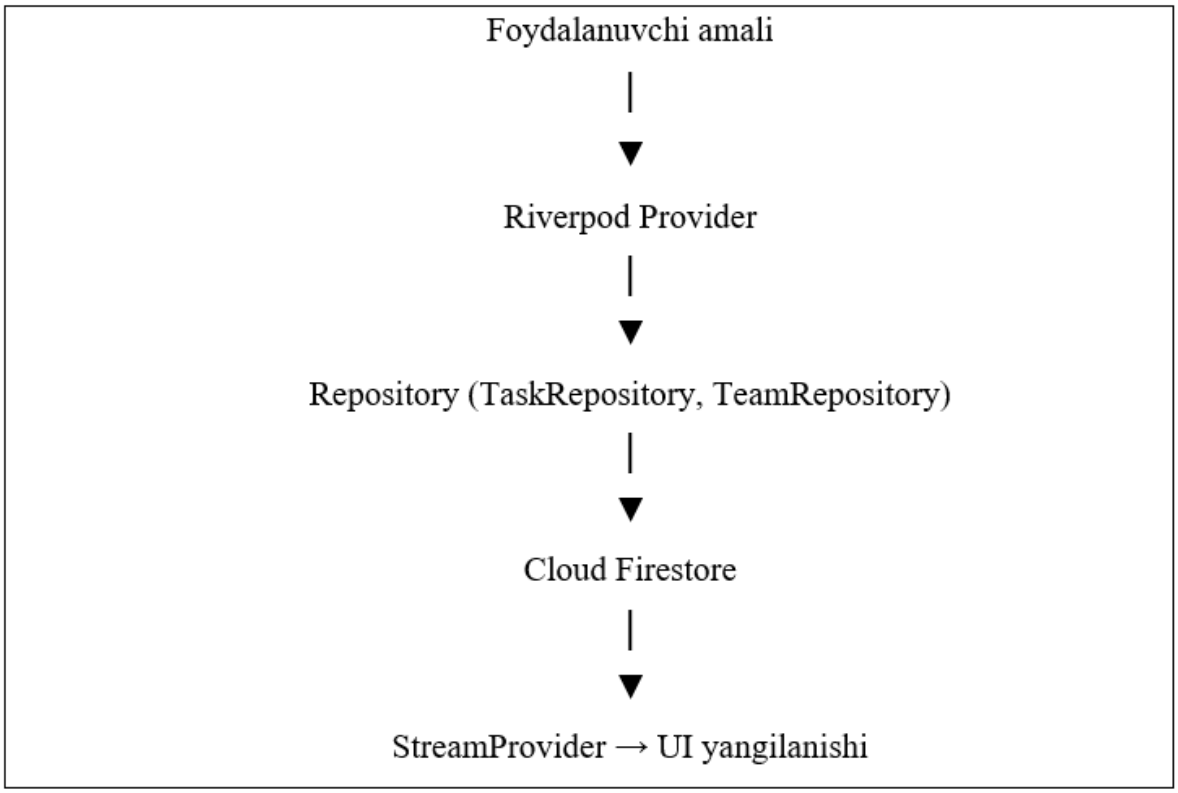


Figure 2. Database

During database integration, data integrity is ensured through Firestore transactions. This guarantees the complete and secure execution of every operation in the system, as well as ensuring that changes made by multiple users at the same time are correctly saved.

Quality control and optimization. Quality control was performed at every technical stage. The cleanliness of the code, the efficiency of Firestore queries, and the usability of the user interface were regularly checked[1].

The system was also tested for speed, fault tolerance, and stability. Image caching via cached_network_image, Riverpod's selective rebuild mechanism, and load states using shimmer were implemented as optimization measures. Issues that arose during the integration process - Firebase token renewal, time zone compatibility of due dates, and biometric authentication across devices - were identified and resolved. In this way, quality control and optimization ensured the

stable operation of the application, a high level of user experience, and system security[2].

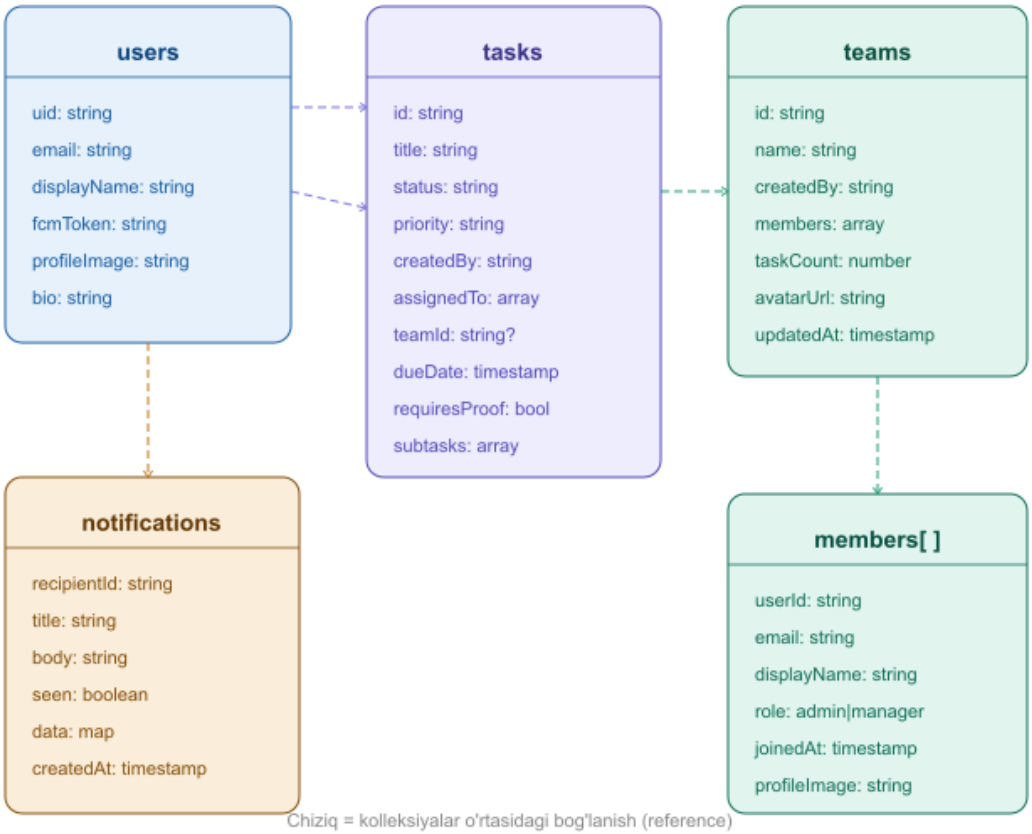


Figure 3. Database in Firestore

User Interface and UX/UI Design. The user interface of an application is crucial to its success. The interface should be user-friendly, intuitive, and fast.

At this stage, the bottom navigation bar (AnimatedBottomNav), filter chips, modal windows, FloatingActionButton, and other interactive components were designed. In addition, the interface design includes visual elements such as a purple primary color, importance level colors, smooth 300 ms animations, and loading states through a shimmer effect[3].

The user interface was tested on Android devices to check for ease of navigation, visual appearance, and functionality. During the process, issues identified were addressed and the interface was further improved.

**References:**

1. Fazylov Sh.Kh. Postroenie modifitsirovannyx algoritmov raspoznavaniya na osnoe potencialnyx funksiy / Sh.K. Fazylov, N.M.Mirzaev, O.N.Mirzaev // Actual problems of appropriate mathematics and information technology - al-Khorezmiy 2009: Trudy mejdunarodnoy conference. - Tashkent, 2009. - T. 2. - P. 66-69.
2. Tuhtanazarov, D., Xodjayeva M., Jumayev T., Mahkamov, A. (2022, June). Computational algorithm and program for determining the indicators of wells based on processing of information of oil fields. In AIP Conference Proceedings (Vol. 2432, No. 1, p. 060021). AIP Publishing LLC.
3. Mahkamov, A. A., Jumayev, T. S., Tuhtanazarov, D. S., Dadamuxamedov, A. I. (2024). Using AdaBoost to improve the performance of simple classifiers. In Artificial Intelligence, Blockchain, Computing and Security Volume 2 (pp. 755-760). CRC Press.
4. Zhumaev, T.S., Mirzaev, N.S., & Makhkamov, A.S. (2015). Algorithms for segmentation of color images based on the selection of strongly coupled elements. Studies of Technical Sciences, (4), 22(27).
5. Matt G. Flutter in Action. – Manning Publications, 2021. – 432 p.
6. Zammetti M. Beginning Flutter: A Hands-On Guide to App Development. – Apress, 2020. – 496 p.
7. Perez F. Flutter Recipes: Mobile Development Solutions. – Packt Publishing, 2021. – 380 p.
8. Roy S. Mobile App Security: Principles and Practices. – Springer, 2020. – 256 p.
9. Nielsen J. Usability Engineering. – Morgan Kaufmann, 2020. – 384 p.
10. Martin R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – Prentice Hall, 2017. – 432 p.
11. Windmill E. Flutter in Practice. – Manning Publications, 2022. – 368 p.
12. Sommerville I. Software Engineering. 10th ed. – Pearson Education, 2016. – 816 p.