Task Manager App Report

Android Application using Kotlin and Jetpack Compose

April 26, 2025

Contents

Chapter 1

Introduction

This report provides a detailed overview of the **Task Manager App**, a mobile application developed using **Kotlin** and **Jetpack Compose**. The app was designed with the goal of supporting teams in managing work shifts, task assignments, and internal communications effectively. Emphasizing simplicity, responsiveness, and modern user interface principles, the application delivers a straightforward solution for everyday team coordination challenges.

Chapter 2

Project Description

The **Task Manager App** was conceived to solve three major problems that small teams often face:

1. Keeping track of weekly work shifts.

2. Managing daily and weekly task assignments.

3. Facilitating instant communication among team members.

Through a clean tabbed interface, the app offers users access to these three major functionalities. Each functionality is built using modern composable principles, making the application lightweight and responsive even on lower-end devices.

Chapter 3

Features

The application's core is divided into three main functional sections, each mapped to a separate tab for intuitive navigation.

3.1 Shifts Tab

This tab displays the **weekly shift schedule** for the team. For each day of the week, assigned team members are listed along with their respective work hours where applicable. The data is presented clearly using a `LazyColumn`, ensuring smooth scrolling even if the schedule grows.

3.2 Tasks Tab

The **Tasks Tab** outlines task assignments for each team member. Examples include roles like *Kitchen Prep*, *Table Service*, and *Inventory Management*. This structured view ensures team members are aware of their individual responsibilities on a daily or weekly basis.

3.3 Chat Tab

The **Chat Tab** integrates a simple yet effective communication system within the app. Users can send and receive short text messages, enabling real-time coordination without switching to an external messaging platform. The chat messages are displayed in a vertically scrollable list, and new inputs are instantly appended for real-time feedback.

Chapter 4

System Architecture

The application follows a modular and reactive architecture, ensuring code maintainability and user experience consistency.

4.1 MainActivity

The `MainActivity` class serves as the entry point of the application. It initializes the main content view using the `setContent` function and renders the `TaskManagerApp()` composable.

4.2 TaskManagerApp Composable

This central composable maintains the currently selected tab's state and updates the UI reactively based on user interactions. It displays the appropriate screen by using a `when` statement linked to the selected tab index.

4.3 Screens as Composable Functions

Each major functionality (Shifts, Tasks, and Chat) is implemented as an independent composable function:

- `ShiftScheduleScreen()`: Displays a list of weekly shifts.
- `TaskAssignmentScreen()`: Displays the list of assigned tasks.
- `ChatScreen()`: Handles messaging, text input, and message listing.

This separation promotes clarity, scalability, and easy future maintenance.

Chapter 5

User Interface Design

The UI design principles center on simplicity, user-friendliness, and responsiveness.

5.1 Layout Components

- `Column` layouts are used for vertical stacking of elements.
- `Row` layouts are used for horizontal groupings, such as the text field and send button in the Chat screen.
- `LazyColumn` is used to efficiently render lists of dynamic data like shifts, tasks, and chat messages.

5.2 Navigation

Navigation within the app is handled through a `TabRow`, allowing users to switch between Shifts, Tasks, and Chat with a single tap. This creates a smooth and intuitive navigation flow without needing complex routing.

5.3 Styling

The design maintains consistent padding, font sizes, and colors. For instance, a subtle grey background highlights chat messages, while titles use a slightly larger font size (20sp) to distinguish them from content. These stylistic choices enhance the app's readability and aesthetic appeal.

Chapter 6

Code Structure and Best Practices

The application code adheres to several best practices:

6.1 State Management

State management is performed using `remember` and `mutableStateOf`, enabling Compose to automatically recompose affected areas when data changes. This ensures a reactive and dynamic user interface without manual refreshes.

6.2 Composable Functions

Breaking down the app into multiple composable functions improves modularity. Each screen is independently maintainable, and updates to one feature do not interfere with others.

6.3 Scalability

The current static data (for shifts, tasks, and chat) can easily be replaced with dynamic data from a database or network call. Thus, the app is designed to be extensible without requiring a complete redesign.

Chapter 7

Potential Improvements

While the current version effectively demonstrates the intended features, several improvements could be made:

- Integrate real-time databases such as Firebase to store shifts, tasks, and chat messages persistently.
- Implement user authentication for secure access and role-based permissions.
- Enhance the Chat feature with timestamps, user names, and typing indicators.
- Add notifications for new messages or shift updates.

Such enhancements would further strengthen the application's utility and make it suitable for larger teams.

Chapter 8

Conclusion

The **Task Manager App** represents a successful implementation of a team management tool built with modern Android development techniques. Using Kotlin and Jetpack Compose, the app offers an intuitive interface, essential features, and scalable architecture. By focusing on usability, clean UI design, and modular code structure, the project lays the groundwork for more advanced applications.

Future developments could transform this simple app into a robust platform for team collaboration. Overall, the project highlights the power and flexibility of Compose and Kotlin for building efficient mobile applications.