Flutter Developer Task Submission

Softrack - Software Deployment Management App

Project Repository GitHub Repository Link (https://github.com/Sumitdubey2255/Softrack)

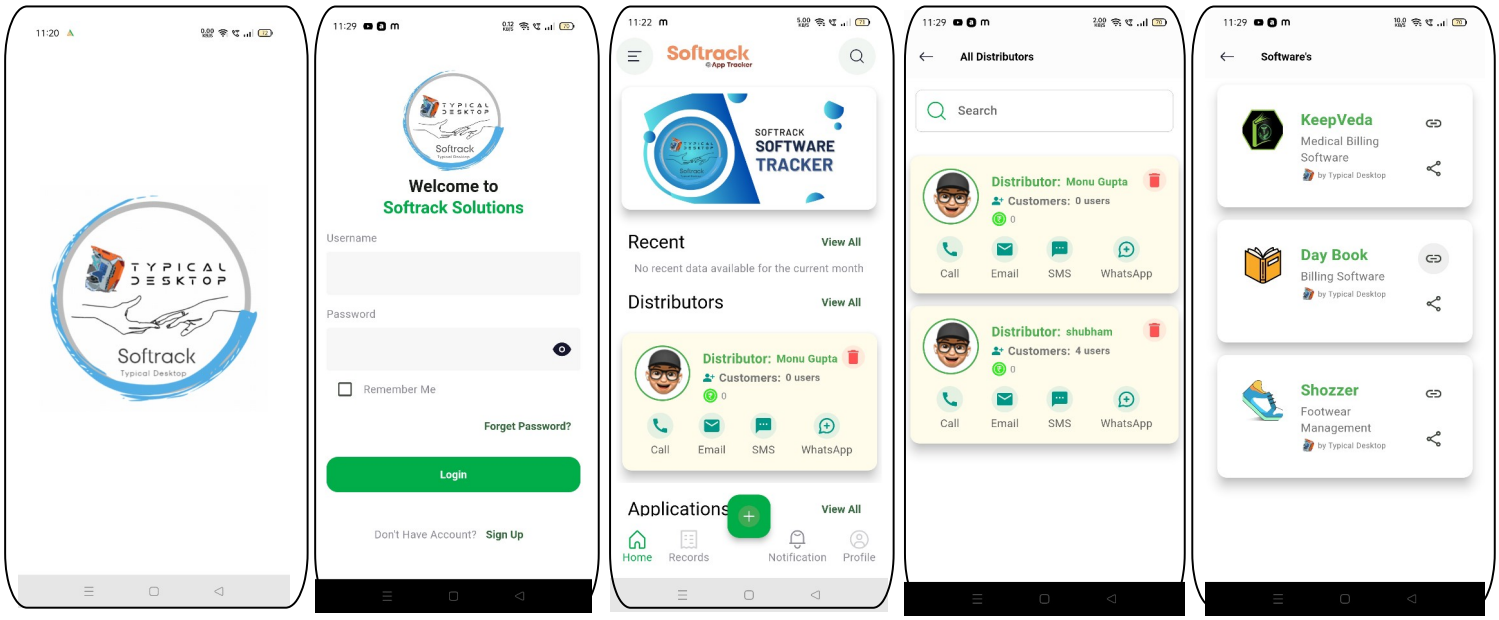
Personal Portfolio Portfolio Link (https://sumit-portfolio-4mn0.onrender.com/)

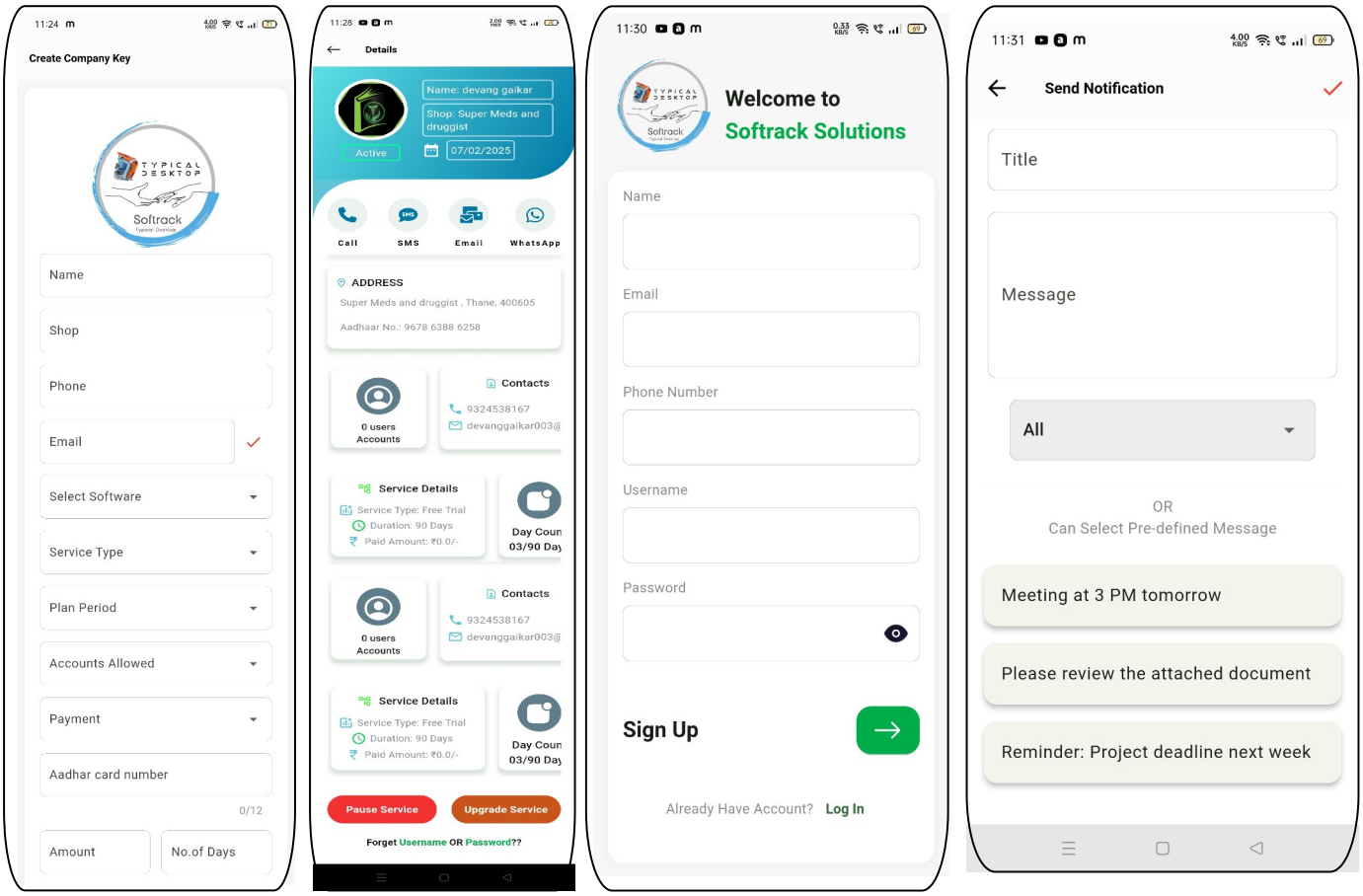
What problem does the app solve?

Softrack addresses the complex challenge of software license management and deployment for businesses with multi-tiered distribution models. The application solves several key problems:

1. **License Management Complexity**: Software vendors often struggle with managing licenses across different distribution channels. Softrack provides a unified platform where admins can generate, distribute, and monitor license keys.
2. **Distribution Hierarchy Management**: The app implements a structured hierarchy with administrators and distributors, allowing for proper access control and permissions while maintaining oversight.
3. **Communication Gaps**: The built-in notification system bridges communication gaps between software providers and end-users, ensuring important updates reach users directly through their licensed software.
4. **Manual License Distribution**: The app automates the license generation and distribution process with email integration, eliminating error-prone manual processes.
5. **Software Version Control**: Softrack enables administrators to manage software updates and new versions in a centralized location, ensuring users always have access to the latest versions.

App Screenshots





Unexpected challenges faced during development

1. Complex State Management

One of the most significant challenges was implementing state management across the app's multi-tiered user hierarchy. Ensuring that administrators and distributors had access to the right data without compromising security required careful planning.

Solution: I implemented a custom state management approach using Provider combined with repository patterns. This allowed for precise control over data flow while maintaining separation of concerns. I created distinct service classes for each user role, ensuring data was properly filtered based on user permissions.

2. Real-time License Validation

Initially, I underestimated the complexity of implementing real-time license validation that would work even with intermittent internet connectivity.

Solution: I developed a hybrid approach that cached validation tokens locally while performing periodic checks with the server. This ensured software could continue functioning during brief connectivity issues while maintaining security through regular validation cycles.

3. Email Delivery Reliability

Email delivery for license keys and OTP verification was surprisingly challenging due to varying email server configurations and spam filtering.

Solution: I implemented a failover system with multiple email service providers and added retry logic with exponential backoff. Additionally, I created a dashboard for administrators to monitor email delivery status and manually resend emails if needed.

4. PostgreSQL Performance with REST API

As the database grew, certain queries became inefficient, especially when filtering license data across the distribution hierarchy.

Solution: I optimized the database schema by implementing proper indexing strategies and rewrote several critical queries. I also implemented caching mechanisms for frequently accessed data and pagination for large result sets to improve overall performance.

What would I do differently if starting the project today?

1. Architecture Improvements

I would adopt a more robust architecture from the beginning. Specifically, I would implement Clean Architecture principles with clear separation between data, domain, and presentation layers. This would make the codebase more maintainable and testable.

2. Testing Strategy

I would implement a comprehensive testing strategy from day one, including unit tests, widget tests, and integration tests. This would have caught several issues earlier in development and reduced debugging time.

3. Backend as a Service Utilization

While Supabase served the project well, I would explore more fully utilizing its real-time capabilities through subscriptions rather than relying heavily on REST API calls. This would improve the app's responsiveness and reduce server load.

4. UI/UX Refinement

I would invest more time in the initial UI/UX design phase, creating a comprehensive design system with reusable components. This would have ensured a more consistent user experience and faster development cycles.

5. Offline Capabilities

I would build stronger offline capabilities from the start. The current implementation handles intermittent connectivity, but a more robust offline-first approach would better serve users in environments with unreliable internet access.

6. Analytics Integration

I would integrate analytics from the beginning to capture user behavior and pain points. This data would have informed development priorities and feature enhancements based on actual usage patterns.

Technologies Used

- **Frontend**: Flutter, Dart
- **Backend**: Supabase, PostgreSQL
- **Authentication**: Custom email OTP verification
- **API**: REST API
- **State Management**: Provider
- **Storage**: Supabase Storage
- **Notifications**: Custom Supabase Messaging Logic
- **Email Service**: Custom email integration

Conclusion

Developing Softrack has been a valuable learning experience in creating a complex B2B application with Flutter. The hierarchical user management, license generation and distribution system, and notification capabilities demonstrate my ability to implement business-critical features in a mobile application. The challenges faced during development have strengthened my problem-solving skills and deepened my understanding of Flutter's capabilities in enterprise applications.