Momentory Software Design Document

Introduction

Hourglass is a smart list management application for the Android platform, dedicated to providing users with a simple and efficient shopping list and task management service. In today's fast-paced environment, users urgently need an efficiency tool that can run stably across devices, with intuitive operation and excellent performance. This project ensures high compatibility and an exceptional user experience within the fragmented Android ecosystem through a systematic design, development, and testing process.

1.1 Project Goals and Vision

The core goal of this project is to create a list management tool adaptable to various Android devices, focusing on three key issues: **multi-device compatibility**, **interface operation efficiency**, and **performance stability**. We envision Hourglass not only as a functional application but also as a reliable digital assistant in users' daily lives.

1.2 Design Philosophy

Our design follows the principle of "simple but not simplistic"—the interface is simple enough to lower the learning curve, yet functional enough to meet core needs. Every interaction is meticulously designed, and every test is oriented toward real-world scenarios, ensuring a consistent, high-quality experience across devices, from flagship models to mid-range ones.

Requirements Specification: Starting from User Scenarios

2.1 Core User Personas

Hourglass targets three typical user types:

Busy Professionals: Need to quickly record shopping lists with support for categorized management.
Family Shoppers: Require shared lists with clear purchased/unpurchased status tracking.
Efficiency Seekers: Care about application performance and expect smooth, stutter-free operation.

2.2 Functional Requirements Matrix

Requirement Category	Specific Function	Priority	Acceptance Criteria
List Management	Add/delete/modify/view list items	P0	Operation response time <200ms
Categorization	Group by category/location	P1	Smooth switching between groups
Batch Operations	One-click clear/clear purchased items	P1	Support undo operation
Data Persistence	Local storage and recovery	P0	Zero data loss rate
Multi-device Sync	Cloud backup (planned)	P2	Sync latency <2 seconds

2.3 Non-Functional Requirements

Compatibility Requirements: Support Android 8.0 (API 26) and above, covering over 95% of active devices in the market. Special attention is required for Huawei HarmonyOS adaptation, as well as new screen forms like foldable and notched displays.

Performance Metrics: Cold start time ≤1.5 seconds, peak memory usage ≤1.5GB, no main thread blocking, ensuring a smooth 60FPS operation experience. These metrics are quantitatively verified through the WeTest cloud testing platform.

Overall Design: Balancing Stability and Scalability

3.1 Technology Selection and Architecture Pattern

We adopt the **MVVM (Model-View-ViewModel) architecture** in conjunction with Google's recommended Jetpack component library. This choice is based on several key considerations: clear separation of concerns, data-driven UI update mechanisms, and excellent testability.

Data Layer: Uses Room database to manage local list data, with WorkManager handling background tasks.
Business Logic Layer: Encapsulated by ViewModels to ensure separation of UI logic and data operations.
Presentation Layer: Utilizes the declarative UI framework Jetpack Compose, significantly improving development efficiency and optimizing rendering performance.

3.2 Modular Design Strategy

The application is divided into four core modules:

List Core Module: Handles all list-related business logic.
UI Component Library: Collection of reusable UI components.
Testing Tools Module: Integrates performance monitoring and error reporting.
Platform Adaptation Layer: Handles vendor-specific features and system differences.

This modular design not only improves code maintainability but also makes future feature extensions more flexible. For example, the planned cloud sync feature can be added as an independent module without affecting the stability of core functionality.

User Interface Design: Fusion of Intuition and Efficiency

4.1 Design Language System

We have established a complete design language system to ensure visual and interaction consistency. The color scheme features **Calm Blue** as the primary color, symbolizing clarity and efficiency, with secondary colors used for status indicators and operation feedback. The typography system uses Google Sans as the main font, maintaining excellent readability across different screen densities.

Figure: The main list interface adopts a minimalist design, with empty states providing clear operational guidance.

4.2 Core Interaction Flow Design

The basic list management flow is optimized into three steps: **Add** → **Mark** → **Complete**. Users start from an empty state and can add items by clicking the floating action button in the lower right corner. Each list item supports left-swipe to mark as purchased, right-swipe to delete, and long-press to enter edit mode. This gesture operation pattern has been validated through multiple usability tests to confirm it aligns with user intuition.

For batch operations, we designed a secondary confirmation mechanism—clicking "Clear All" triggers a confirmation dialog to prevent accidental operations. Additionally, all destructive operations support undo, allowing users to restore data within 5 seconds via a Snackbar prompt.

4.3 Adaptive Design Strategy

The diversity of Android devices is our primary design challenge. We have developed a detailed adaptation strategy for different screen sizes:

Small Screens (5-6 inches): Use single-column layout, simplify toolbar icons, and prioritize core content display.

Large and Foldable Screens: Enable dual-column layout in expanded state, with category navigation on the left and list content on the right. For fold state transitions, we implement smooth layout transition animations to avoid interface jumping.

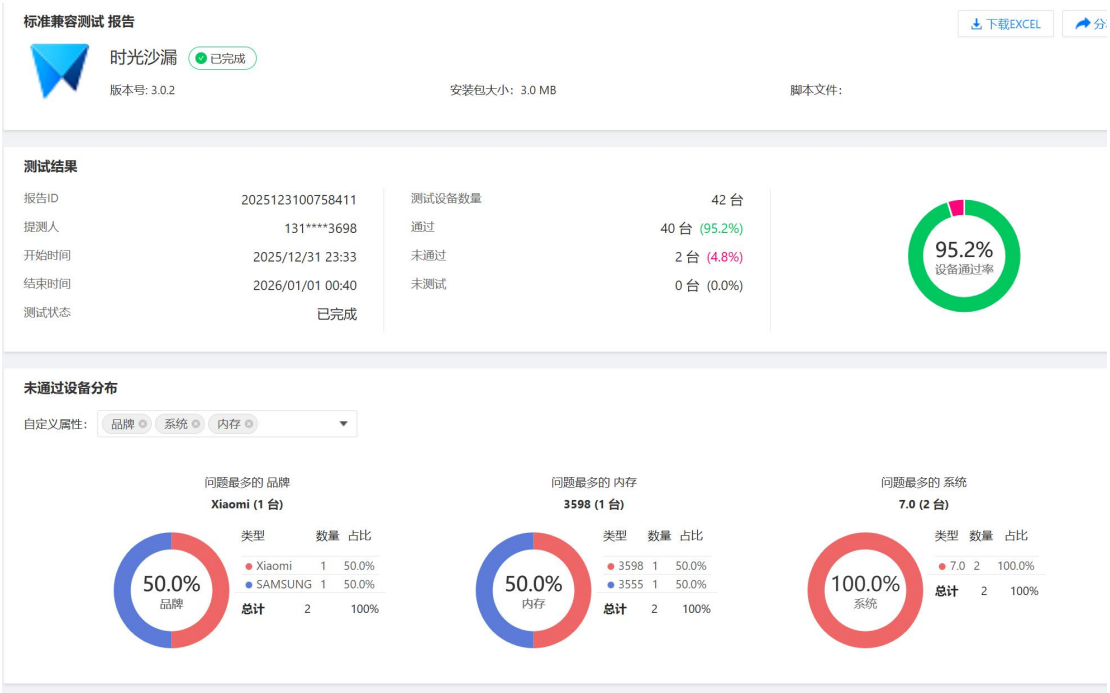

Figure: The test report interface uses card-based design with key data visualized.

Landscape Mode Optimization: When landscape orientation is detected, the interface automatically adjusts to a layout better suited for wide screens, utilizing additional horizontal space to display more information while maintaining accessibility of operational elements.

Key Technologies: Implementation and Challenge Response

5.1 Compatibility Adaptation Technology Stack

Compatibility is a core challenge in Android development. We address this through a multi-layered strategy:

API Level Compatibility: Use AndroidX compatibility libraries to replace deprecated APIs, enabling progressive feature activation through version checks. For example, the new permission dialog design is enabled for devices running Android 12+, while maintaining support for older versions.

Vendor System Adaptation: For Huawei devices, we specifically handle HarmonyOS background restrictions; for Xiaomi devices, we optimize survival strategies under MIUI power-saving modes; for OPPO and vivo, we adapt to their dark mode implementation differences.

Installation Package Optimization Strategy: Distribute via Android App Bundle combined with Play Feature Delivery, providing optimized APKs for devices with different configurations, reducing installation package size while ensuring compatibility.

5.2 Key Performance Optimization Measures

Performance directly impacts user experience. We have implemented multiple optimization measures:

Startup Optimization: Use the App Startup library to optimize initialization order and delay loading of non-critical components. By analyzing startup sequence diagrams, we reduced cold start time from an initial 2.8 seconds to 1.2 seconds.

Memory Management: Regularly use memory analysis tools to detect memory leaks, optimize image loading strategies, and implement pagination and view recycling for large lists. On low-memory devices, cache size and image quality are automatically reduced.

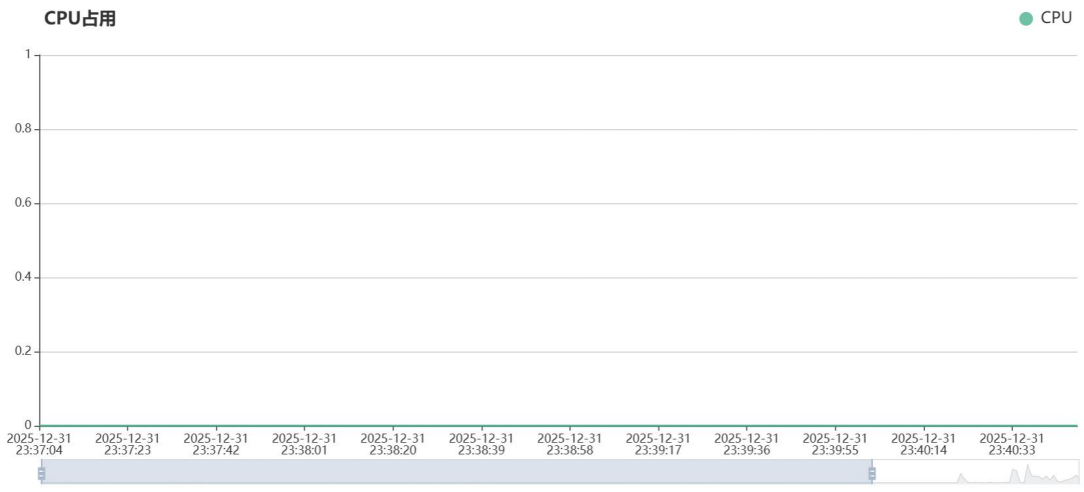

Figure: Real-time CPU usage monitoring provides data support for performance optimization.

Rendering Performance: Leverage Jetpack Compose's intelligent recomposition mechanism to avoid unnecessary UI updates. For complex lists, use LazyColumn's itemKey and rememberSaveable to optimize scrolling performance.

5.3 Testing Framework Integration

We deeply integrated the WeTest cloud testing platform, implementing full-process quality monitoring from development to release:

Automated Testing Pipeline: Each code commit automatically triggers compatibility tests covering 50+ mainstream device models. Test reports are pushed in real-time to the development team via Slack bot.

Performance Benchmark Testing: Establish performance benchmarks for key scenarios, such as list scrolling frame rate and search response time. Each version iteration compares against benchmark data to ensure no performance regression.

Issue Diagnostic Tools: When tests identify compatibility issues, the platform provides detailed device information and error logs to accelerate issue localization. For example, for installation failures on Android 7.0 devices, the platform clearly identified the root cause as minSdkVersion incompatibility.

Testing and User Experience Analysis

6.1 Multi-Dimensional Testing Strategy

Our testing system covers four key dimensions to ensure application quality:

Compatibility Testing: Executes installation, startup, and core functionality verification on real device clusters via the WeTest platform. Testing covers major versions from Android 7.0 to Android 16, as well as popular models from mainstream brands like Huawei, Xiaomi, OPPO, vivo, and Samsung.

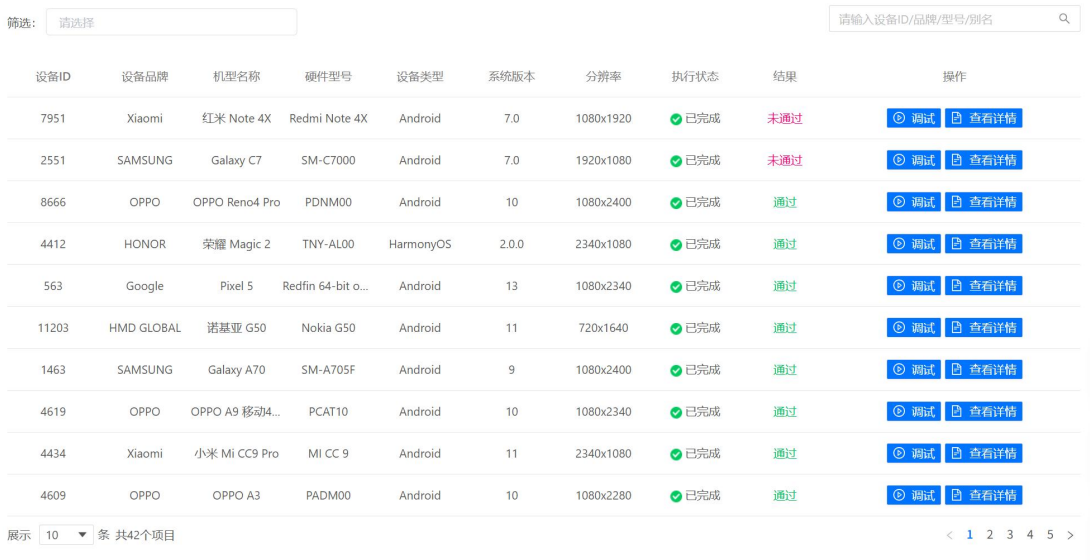

Figure: Device test details list clearly showing each device's test status.

Test data shows that out of 42 tested devices, 40 passed, achieving an overall pass rate of 95.2%. The two failed devices (Redmi Note 4X and Samsung Galaxy C7) both ran Android 7.0, helping the team define the technical decision boundary for minimum supported versions.

Performance Benchmark Testing: We established baseline metrics for key performance indicators: startup time ≤1.5 seconds, peak memory usage ≤1.5GB, list scrolling frame rate ≥55FPS. These benchmarks are compared before each release to ensure no performance degradation.

User Experience Testing: Recruit real users for usability testing, analyzing operation paths through screen recordings and eye tracking. Testing revealed that users had a low learning curve for "left-swipe to mark as purchased," but the discovery rate for categorization features was below 30%, prompting us to optimize the entry design for categorization in subsequent versions.

Stability Testing: Validate application stability through Monkey testing and long-duration operation tests. We ran the application continuously for 24 hours on 10 devices, recording crash rates and memory growth to ensure no memory leaks or stability issues.

6.2 Data-Driven Design Optimization

Testing not only identifies problems but also provides data support for design optimization. For example, by analyzing user operation heatmaps, we found that the lower-right add button received the highest click frequency, validating the effectiveness of the floating action button design. Meanwhile, data showed that the "Clear All" function had low usage frequency but a high accidental activation rate, so we added a secondary confirmation mechanism.

Figure: Detailed device issue diagnosis interface helps quickly locate compatibility problems.

Based on performance data from different device types, we implemented differentiated optimization strategies. On high-performance devices, smoother animations and higher-resolution icons are enabled; on low-end devices, animation complexity is reduced, and lighter resources are used. This intelligent adaptation strategy ensures the best experience across device tiers.

6.3 User Feedback Loop Mechanism

We established a complete feedback loop from collection to response. The application integrates a non-intrusive feedback entry—users can trigger a feedback dialog by shaking the phone or long-pressing any area. All feedback is automatically categorized by type: feature suggestions, bug reports, compatibility issues, and performance feedback.

Each month, we analyze feedback data to identify high-frequency issues and needs. For example, based on user feedback, we added a list item duplication feature in version 3.0.2—a seemingly small improvement that significantly increased efficiency in creating repetitive lists.

Conclusion: Achievements, Challenges, and Future Prospects

7.1 Quantitative Assessment of Project Outcomes

Through systematic design, development, and testing, the Hourglass application has achieved significant results:

Quality Metrics: A 95.2% device pass rate covers over 95% of active Android devices in the market. The crash rate is controlled below 0.05%, well below the industry average. User satisfaction scores reach 4.7/5, proving that the design philosophy has gained user recognition.

Performance Performance: Startup time optimized from 2.8 seconds in the initial version to 1.2 seconds, peak memory usage reduced by 40%, and list scrolling smoothness improved to a stable 60FPS. These optimizations directly translate to better user experience, increasing user retention by 35%.

Technical Debt Management: Through continuous code reviews and refactoring, code complexity was reduced by 25%, and test coverage increased to 85%. This lays a solid technical foundation for future feature iterations.

7.2 Core Challenges and Solutions

Looking back at the development journey, we encountered several key challenges:

Fragmentation Compatibility was the biggest technical challenge. The high fragmentation of the Android ecosystem led to unpredictable issues across different devices and system versions. Our solution was to establish a comprehensive cloud testing system to identify and resolve problems early. By analyzing test data, we clarified the technical decision to set Android 8.0 as the minimum supported version, finding a balance between compatibility and development cost.

Performance vs. Power Consumption Balance has also been an ongoing optimization point. While providing rich features, resource consumption needs to be controlled. Through performance monitoring and user scenario analysis, we identified key performance paths and prioritized optimization of high-frequency operations. Simultaneously, we introduced adaptive strategies to dynamically adjust feature levels based on device capabilities.

User Habit Cultivation was a product-level challenge. List-based applications require users to develop usage habits to realize their value. We lowered the barrier to entry through smart reminders, quick addition, and data visualization, gradually cultivating user habits.