Project: SipSip—Plant Care Mobile Application
Course: COMP 3059—Capstone Project
Team Members: Jayden Lewis, Uzma Shaikh, Aidan Repchik
Instructor: Anjana Shah
Date: November 19, 2025

SipSip—Technology Requirements Document

PART II - Technology Requirements

1.0 Technological Requirements

The following table outlines the core technologies planned for the development of the SipSip Android mobile application. Each technology has been selected to meet project constraints (zero budget, Android-only platform, two-semester timeline) and deliver a functional MVP by March 27, 2026.

Technology Category	Technology Selected	Pros and Cons
Programming Language	Kotlin	Pros: Modern, concise syntax with null safety features; Official language for Android development recommended by Google; Strong interoperability with Java libraries; Excellent coroutines support for asynchronous operations; Large community and extensive documentation. Cons: Slightly longer compilation time compared to Java; Team members have varying levels of Kotlin experience requiring initial learning curve.
Mobile Framework	Android SDK (API Level 35, backward compatible to API 24)	Pros: Native Android development ensures optimal performance; Full access to device features (camera, notifications, local storage); Meets Google Play's target API requirements; Supports Android 7.1 Nougat and above covering 95%+ of active devices; Free and well-documented. Cons: Android-only (no iOS or cross-platform support); Requires understanding of Android lifecycle and components; Device fragmentation requires extensive testing across different screen sizes and OS versions.

Architecture Pattern	MVVM + Clean Architecture	Pros: Separation of concerns improves code maintainability and testability; ViewModel survives configuration changes; Clear data flow from UI to data layer; Aligns with Android best practices; Facilitates team collaboration with defined layer responsibilities; Supports offline-first approach. Cons: Initial setup complexity and boilerplate code; Requires team understanding of architectural principles; May be over-engineered for simple features; Learning curve for team members new to Clean Architecture.
UI Framework	Jetpack Compose + Material Design 3	Pros: Modern declarative UI toolkit simplifying UI development; Reduces boilerplate code compared to XML layouts; Real-time preview in Android Studio accelerates development; Better state management and reactivity; Officially recommended by Google for new Android projects; Material Design 3 provides playful, modern components aligned with SipSip brand. Cons: Relatively newer framework requiring learning investment; Some team members lack prior Compose experience; Limited availability of third-party libraries compared to traditional View system; Material Design 3 theming requires additional configuration.
Navigation	Jetpack Navigation Component	Pros: Type-safe navigation with compile-time checking; Handles fragment transactions automatically; Deep linking support; Backstack management simplified; Integrates seamlessly with Jetpack Compose; Well-documented and actively maintained. Cons: Learning curve for navigation graph setup; Complex nested navigation can be challenging; Requires careful planning of navigation structure upfront.
Local Database	Room Persistence Library	Pros: Provides abstraction layer over SQLite for robust database access; Compile-time verification of SQL queries reducing runtime errors; Seamless integration with Android architecture components and Kotlin Flow; Supports offline-first functionality critical for SipSip (S16); Excellent documentation and community support; Free and lightweight. Cons: Requires understanding of database schema design and migrations; Setup involves creating entities,

		DAOs, and database classes; May have performance limitations with extremely large datasets (not anticipated for MVP scope).
Dependency Injection	Hilt (Dagger wrapper)	Pros: Simplifies dependency injection setup compared to pure Dagger; Compile-time dependency graph validation; Reduces boilerplate with annotations; Integrates seamlessly with Android components; Promotes testability and modularity; Official Google recommendation. Cons: Additional build time overhead; Complex error messages during setup; Requires understanding of DI principles; Overkill for very small projects but beneficial at SipSip's scale.
Background Processing	WorkManager + AlarmManager	Pros: WorkManager handles deferrable background tasks with guaranteed execution; Respects battery optimization; AlarmManager provides exact alarm scheduling for critical reminders (S04, S13); Works offline; Handles device restarts gracefully; Integrated with Android system constraints. Cons: AlarmManager requires SCHEDULE_EXACT_ALARM permission on Android 12+; WorkManager constraints may delay non-urgent tasks; Complex setup for coordinated background work; Testing background tasks requires specific tooling.
Image Loading	Coil	Pros: Kotlin-first image loading library; Lightweight and fast; Coroutines support; Memory and disk caching built-in; Jetpack Compose integration; Handles plant profile photos efficiently. Cons: Smaller community compared to Glide/Picasso; Fewer advanced features; Relatively newer library.
Cloud Database (Post-MVP)	Supabase	Pros: Free tier offers 1GB storage and 5GB data transfer suitable for household sharing feature (S09); Built on PostgreSQL providing relational database capabilities; Real-time subscriptions for collaborative features; RESTful API and client libraries for easy integration; Authentication and security features built-in. Cons: Requires internet connection for synchronization; Free tier limitations may require upgrade if user base grows; Dependency on third-party service introduces

		reliability concerns; Not required for MVP, planned for post-MVP household sync feature.
Image Recognition API	Google Cloud Vision API	Pros: Highly accurate plant identification using machine learning (S02); Free tier includes 1,000 requests per month sufficient for testing and early adoption; Official Google service with reliable uptime; Simple REST API integration; Returns confidence scores for identification results. Cons: Requires internet connection for plant identification; API key management and security considerations; Rate limiting on free tier may restrict heavy usage; Cost implications if usage exceeds free tier in future.
Weather API	Open-Meteo API	Pros: Completely free with no API key required; Provides current weather and 7-day forecasts; No rate limits on free tier; Open-source and privacy-focused; Supports location-based weather retrieval; Enhances outdoor plant care recommendations (S05). Cons: Requires internet connection for weather updates; Less feature-rich than commercial alternatives; Weather data accuracy varies by region; Not critical for MVP core functionality; Community-maintained service may have reliability concerns.
Version Control	GitHub	Pros: Industry-standard version control system; Free for public and private repositories; Excellent collaboration features (pull requests, code reviews, issue tracking); Integration with Android Studio; Team already familiar with Git workflows; Enables backup and disaster recovery; GitHub Classroom integration for instructor review. Cons: Requires discipline in branching strategy and merge conflict resolution; Learning curve for team members less experienced with Git; Potential for merge conflicts in collaborative development; Sensitive data (API keys) must be excluded via .gitignore.

Development IDE	Android Studio	Pros: Official IDE for Android development with full SDK integration; Built-in emulator for testing without physical devices; Advanced debugging and profiling tools; Jetpack Compose preview and UI inspector; Free and cross-platform (Windows, macOS, Linux); Regular updates with new features and performance improvements. Cons: Resource-intensive requiring powerful development machines; Slow build times on older hardware; Large installation size (multiple GB); Occasional stability issues with beta features; Gradle sync can be slow.
Build System	Gradle	Pros: Default build system for Android projects; Flexible dependency management; Supports incremental builds reducing compilation time; Extensible with custom build logic; Industry standard with extensive documentation; Kotlin DSL support improves readability. Cons: Build times can be slow for large projects; Complex Gradle scripts can be difficult to debug; Version compatibility issues between Gradle, Android Gradle Plugin, and Kotlin; Memory-intensive during builds.
Testing Framework	JUnit 5, MockK, Turbine, Compose Test	Pros: JUnit 5 is standard for unit testing in Android; MockK provides Kotlin-friendly mocking; Turbine simplifies Flow testing; Compose Test enables UI testing for Jetpack Compose; Integration with Android Studio for automated test execution; Supports test-driven development; Free and open-source. Cons: Writing comprehensive tests requires additional development time; UI tests can be flaky due to timing issues; Team has limited experience with automated testing; Maintaining test suites adds overhead.
Hardware (Development)	Personal Laptops & Android Devices	Pros: Team members already own development hardware; No additional costs; Physical Android devices (API 24-35) provide realistic testing environment; Android Studio emulator available as backup; Covers target device range. Cons: Hardware performance varies across team members; Limited device variety for testing

		fragmentation; No access to specialized testing labs or device farms; Potential for hardware failures without backup; Emulator performance depends on host machine specs.
Design & Prototyping	Figma	Pros: Free tier adequate for student projects; Collaborative design platform for UI/UX mockups; Export assets for Android development; Component libraries for Material Design 3; Real-time collaboration and feedback; Interactive prototyping capabilities required for Sprint 4. Cons: Requires internet connection; Learning curve for design principles; Export formats may require manual adjustments for Jetpack Compose; Free tier limits number of projects and collaborators.

Table 1: Technological Requirements for SipSip Application

2.0 Learning Plan

The following table outlines the technical skills required for SipSip development, each team member's current proficiency level, assigned responsibilities, and detailed learning plans to bridge skill gaps before and during development.
Note on Feature References: S01-S16 refer to approved MVP features listed in SipSip - Requirements Analysis & Design.docx, Section 1.2 Scope.

Technical Skill	Team Member	Current Skill Level	Responsibility & Learning Plan
Kotlin + Coroutines + Flow	Jayden Lewis	85%	Responsibilities: Leads dashboard implementation (S03), navigation logic, reminder scheduling coordination (S04), and activity logging (S10). Learning Plan: Start Date: November 25, 2025 End Date: December 15, 2025 Resources: Kotlin Koans exercises, Coroutines official guide, Kotlin Flow documentation

			Activities: Complete remaining Kotlin Koans, practice Flow operators, mentor team on coroutines best practices Target: 90% proficiency
	Uzma Shaikh	80%	Responsibilities: Builds reminder notification screens (S04, S13), notification settings UI (S07), and shared household interface components (S09). Learning Plan: Start Date: November 20, 2025 End Date: December 10, 2025 Resources: Kotlin official tutorials, team pair programming sessions Activities: Daily practice with pair programming, focus on coroutines for background tasks Target: 85% proficiency
	Aidan Repchik	88%	Responsibilities: Leads plant profile CRUD operations (S01), data model architecture, and Google Cloud Vision API integration (S02). Learning Plan: Start Date: Ongoing End Date: Ongoing Resources: Code review participation Activities: Focus on code reviews and mentoring team members, maintain current proficiency Target: Maintain 88% proficiency
Jetpack Compose (layouts, animations, theming)	Jayden Lewis	85%	Responsibilities: Leads dashboard composables (S03), implements playful animations, establishes overall Material Design 3 theme, creates reusable UI components. Learning Plan: Start Date: November 20, 2025 End Date: December 20, 2025 Resources: Jetpack Compose Academy codelabs, Material Design 3 guidelines, official documentation

			Activities: Complete advanced Compose codelabs, mentor team members, establish theme and component library Target: 90% proficiency
	Uzma Shaikh	75%	Responsibilities: Creates reminder summary cards (S04), tip of day screens (S06), do not disturb settings interface (S07). Learning Plan: Start Date: November 25, 2025 End Date: December 20, 2025 Resources: Compose basics pathway, weekly pairing with Jayden Activities: Complete Compose basics pathway, weekly pair programming sessions with Jayden, practice state management Target: 85% proficiency
	Aidan Repchik	82%	Responsibilities: Builds plant profile creation and editing forms (S01), image picker integration, plant detail screens. Learning Plan: Start Date: November 25, 2025 End Date: December 15, 2025 Resources: Compose forms documentation, Material Design input components Activities: Study form validation patterns, support Uzma with complex layouts Target: 88% proficiency
Room Persistence Library + Flow	Jayden Lewis	80%	Responsibilities: Implements activity log storage (S10), offline data caching strategies, participates in database schema reviews. Learning Plan: Start Date: November 30, 2025 End Date: December 15, 2025 Resources: Room with Flow codelab, Android data layer guide Activities: Complete Room with Flow codelab, implement activity log DAOs, review database migrations

			Target: 85% proficiency
	Uzma Shaikh	78%	Responsibilities: Stores user preferences, reminder data and schedules, notification settings persistence. Learning Plan: Start Date: November 25, 2025 End Date: December 10, 2025 Resources: Official Room codelab, Android developer documentation Activities: Complete official Room codelab, implement preference storage, practice database queries Target: 85% proficiency
	Aidan Repchik	92%	Responsibilities: Leads complete Room database schema design, plant profile entities and DAOs (S01), reminder tables, database migrations, mentors team on Room best practices. Learning Plan: Start Date: Ongoing End Date: Ongoing Resources: Mentoring and code reviews Activities: Design normalized schema, create migration strategies, conduct code reviews, mentor team members Target: Maintain 92% proficiency
Hilt / Dependency Injection	Jayden Lewis	85%	Responsibilities: Configures shared application modules, implements ViewModels with DI, reviews injection setup. Learning Plan: Start Date: December 1, 2025 End Date: December 15, 2025 Resources: Hilt documentation, code review sessions Activities: Participate in architecture reviews, ensure consistent DI patterns Target: Maintain 85% proficiency

	Uzma Shaikh	75%	Responsibilities: Implements dependency injection for feature modules, notification managers, preference repositories. Learning Plan: Start Date: November 28, 2025 End Date: November 30, 2025 Resources: Official Hilt codelab, Android DI guide Activities: Complete official Hilt codelab by November 30, practice with feature module injection Target: 85% proficiency
	Aidan Repchik	92%	Responsibilities: Leads entire Hilt configuration and setup across the project, establishes DI architecture patterns, creates modules for data and domain layers. Learning Plan: Start Date: Ongoing End Date: Ongoing Resources: Architecture planning and mentoring Activities: Design DI graph structure, configure Hilt modules, mentor team on DI principles Target: Maintain 92% proficiency
WorkManager + AlarmManager (Exact Alarms)	Uzma Shaikh	88%	Responsibilities: Leads all reminder scheduling logic (S04), snooze functionality (S13), do not disturb rules implementation (S07), handles Android 12+ exact alarm permissions. Learning Plan: Start Date: November 25, 2025 End Date: December 10, 2025 Resources: WorkManager guide, AlarmManager documentation, Android 14+ exact alarm policies Activities: Implement reminder workers, configure exact alarms with proper permissions, document Android 14+ exact alarm rationale, test on physical devices Target: 93% proficiency

	Jayden Lewis & Aidan Repchik	78%	Responsibilities: Support Uzma with testing, provide backup implementation assistance, test on physical devices across different Android versions. Learning Plan: Start Date: December 1, 2025 End Date: December 15, 2025 Resources: Testing documentation, device testing Activities: Test reminder scheduling on physical devices (API 24-35), verify do not disturb behavior, assist with edge case handling Target: 82% proficiency
Google Cloud Vision API	Jayden Lewis	90%	Responsibilities: Leads API integration (S02), implements error handling and retry logic, manages auto-fill functionality for plant identification, handles API key security. Learning Plan: Start Date: December 5, 2025 End Date: December 20, 2025 Resources: Google Cloud Vision API documentation, Android security best practices Activities: Implement secure API key storage, create plant identification service, handle network errors gracefully, optimize API usage Target: 95% proficiency
	Aidan Repchik	85%	Responsibilities: Provides backup implementation support, assists with testing, handles image preprocessing for optimal API results. Learning Plan: Start Date: December 10, 2025 End Date: December 20, 2025 Resources: Vision API documentation, testing support

			Activities: Test plant identification accuracy, assist with error handling, optimize image quality before API calls Target: 88% proficiency
Open-Meteo API & Weather Logic	Aidan Repchik	88%	Responsibilities: Leads outdoor plant watering schedule adjustments based on weather (S05), implements weather alert notifications, handles location permissions. Learning Plan: Start Date: December 5, 2025 End Date: December 15, 2025 Resources: Open-Meteo API documentation, Android location services guide Activities: Integrate weather API, implement precipitation-based watering logic, create weather alert system Target: 92% proficiency
	Uzma Shaikh	70%	Responsibilities: Implements offline fallback behavior when weather data unavailable, caches weather forecasts locally. Learning Plan: Start Date: December 1, 2025 End Date: December 10, 2025 Resources: API documentation, offline caching patterns Activities: Study offline first patterns, implement weather data caching, handle network unavailability gracefully Target: 80% proficiency
Supabase (Realtime + Auth) - Post-MVP	Aidan Repchik	82%	Responsibilities: Leads prototype development for future household sharing synchronization (S09), establishes authentication patterns, designs real-time data sync strategy. Learning Plan: Start Date: December 15, 2025 End Date: December 20, 2025

			Resources: Supabase official documentation, Kotlin client library tutorials Activities: Complete Supabase tutorials, create proof of concept for household sync, document integration approach for future sprints Target: 88% proficiency
	Jayden Lewis & Uzma Shaikh	68%	Responsibilities: Study client side integration patterns, prepare for future household sync implementation when Aidan's prototype is ready. Learning Plan: Start Date: January 2026 End Date: February 2026 Resources: Supabase documentation, prototype codebase Activities: Review Aidan's prototype, understand Supabase client patterns, plan integration with existing local database Target: 75% proficiency
Testing (JUnit 5, MockK, Turbine, Compose Test)	Jayden Lewis	85%	Responsibilities: Leads unit tests for dashboard (S03) and reminder logic (S04), implements Compose UI tests, establishes testing patterns and standards, targets 80% code coverage. Learning Plan: Start Date: December 1, 2025 End Date: December 20, 2025 Resources: JUnit 5 documentation, MockK guide, Compose testing codelab Activities: Create test templates, write comprehensive unit tests, implement UI tests for critical user flows, mentor team on testing best practices Target: 90% proficiency

	Aidan Repchik	83%	Responsibilities: Focuses on data layer and repository tests, implements DAO tests, tests database migrations. Learning Plan: Start Date: December 5, 2025 End Date: December 20, 2025 Resources: Room testing guide, repository testing patterns Activities: Write comprehensive DAO tests, verify database migrations, test repository logic with MockK Target: 88% proficiency
	Uzma Shaikh	75%	Responsibilities: Writes instrumentation tests for reminder notifications, assists with manual device testing across multiple Android versions, documents test cases. Learning Plan: Start Date: December 1, 2025 End Date: December 20, 2025 Resources: Espresso documentation, instrumentation testing guide, weekly pairing with Jayden Activities: Complete instrumentation testing codelab, write notification tests, perform manual testing on physical devices, weekly pair programming with Jayden Target: 83% proficiency

Table 2: Technical Skills and Learning Plan

3.0 Technology Risk Assessment

While the selected technologies align well with project requirements, the following risks have been identified and mitigation strategies established:

- Learning Curve Risk (Medium)
  Jetpack Compose and Clean Architecture are relatively modern approaches. Team proficiency ranges from 75-88% across key technologies.
  Mitigation: Allocate dedicated learning time (Nov 25 - Jan 15), implement weekly pair programming sessions, complete official codelabs by specified deadlines, leverage team mentors.
- API Dependency Risk (Medium)
  Google Cloud Vision API free tier provides 1,000 requests/month, possibly insufficient during heavy testing.

Mitigation: Local image caching, optimize usage, monitor request limits, implement fallbacks, consider upgrade if required.

- Hardware Limitations (Low)
  Reliance on personal laptops and limited Android devices.
  Mitigation: Utilize emulators, coordinate device testing, document minimum requirements.
- Third-Party Service Reliability (Medium)
  Dependencies on external APIs.
  Mitigation: Comprehensive error handling, offline fallback, cache responses, monitor services, degrade gracefully if APIs unavailable.
- Timeline Pressure (High)
  Two-semester deadline with significant learning required.
  Mitigation: Prioritize MVP features (S01-S10, S16), defer advanced features to post-MVP, maintain strict sprint discipline, build buffer time, front-load learning.
- Build & Compilation Time (Low)
  Gradle builds can be slow with Jetpack Compose/Hilt.
  Mitigation: Optimize Gradle, enable incremental builds/cache, upgrade hardware if needed, modularize code.
- Testing Complexity (Medium)
  Limited prior automated testing experience.
  Mitigation: Start early in Sprint 4, create templates, pair testing leads with others, prioritize critical tests, accept lower initial coverage if needed.

4.0 Technology Approval

Name	Role	Signature	Date
Jayden Lewis	Project Manager	Jayden Lewis	November 23, 2025
Uzma Shaikh	Project Engineer	Uzma Shaikh	November 23, 2025
Aidan Repchik	Project Developer	Aidan Repchik	November 23, 2025

Table 3: Technology Requirements Approval