# Member 1: Infrastructure & Integration Lead - Complete Detailed Prompts

## 🎯 Role Overview

You are responsible for building the foundation: connecting to all external services (Gmail, Slack, Google Sheets, Drive) and setting up the project infrastructure that the entire team will build upon.

---

## 📅 DAY 1: Project Setup & Core Infrastructure

### \*Session 1: Repository & Project Structure (1-2 hours)\*

\*Prompt 1: Initialize Project\*

Create a Node.js TypeScript project with the following structure:

- src/integrations/ for API connections

- src/agents/ for LLM logic (placeholder)

- src/workflows/ for orchestration (placeholder)

- src/utils/ for shared helpers

- src/types/ for TypeScript interfaces

- Include package.json with dependencies: dotenv, @slack/bolt, googleapis, @notionhq/client, axios, winston for logging

- Add tsconfig.json with strict mode enabled

- Create .gitignore that excludes node\_modules, .env, and dist/

\*Prompt 2: Environment Configuration Setup\*

Create a .env.example file with the following structure and add detailed comments for each variable:

- GMAIL\_CLIENT\_ID, GMAIL\_CLIENT\_SECRET, GMAIL\_REDIRECT\_URI

- SLACK\_BOT\_TOKEN, SLACK\_SIGNING\_SECRET, SLACK\_APP\_TOKEN

- GOOGLE\_SHEETS\_API\_KEY

- GOOGLE\_DRIVE\_API\_KEY

- NOTION\_API\_KEY

- TRELLO\_API\_KEY, TRELLO\_TOKEN

- COMPOSIO\_API\_KEY

- LLM\_API\_KEY (for Groq)

- LOG\_LEVEL=info

Also create a src/config/index.ts file that loads and validates all environment variables with helpful error messages if any are missing.

\*Prompt 3: Error Handling & Logging Framework\*

Create a src/utils/logger.ts file using Winston that:

- Logs to console with color-coded levels (error=red, warn=yellow, info=green)

- Logs to files: logs/error.log for errors, logs/combined.log for all logs

- Includes timestamps in ISO format

- Exports different log levels: logger.error(), logger.warn(), logger.info(), logger.debug()

- Creates logs directory if it doesn't exist

\*Prompt 4: Base Integration Class\*

Create a src/integrations/base-integration.ts abstract class that:

- Has methods: connect(), disconnect(), isConnected(), retry()

- Implements exponential backoff retry logic (3 attempts with 1s, 2s, 4s delays)

- Has error handling with try-catch and logging

- Includes connection status tracking

- Uses TypeScript generics for type safety

- Has a rateLimit() method that prevents exceeding API limits

---

### \*Session 2: Gmail Integration (2-3 hours)\*

\*Prompt 5: Gmail Authentication Setup\*

Create src/integrations/gmail/auth.ts that:

- Uses googleapis library for OAuth2 authentication

- Implements OAuth2 flow with credentials from environment variables

- Saves and loads tokens from a local file (tokens/gmail-token.json)

- Handles token refresh automatically

- Exports getAuthenticatedClient() function

- Includes comprehensive error handling with user-friendly messages

- Add JSDoc comments explaining the OAuth flow

\*Prompt 6: Gmail Message Fetcher\*

Create src/integrations/gmail/fetcher.ts that:

- Extends the BaseIntegration class

- Implements fetchUnreadMessages() that gets emails from the last 5 minutes

- Implements fetchMessageById(messageId) for specific emails

- Parses email data into a structured format: { id, from, subject, body, date, labels, attachments[] }

- Handles multipart MIME emails and extracts plain text

- Filters out spam and promotions by checking labels

- Includes pagination for large result sets

- Implements markAsRead(messageId) function

- Has error handling for network failures and invalid responses

\*Prompt 7: Gmail Listener Service\*

Create src/integrations/gmail/listener.ts that:

- Implements a polling mechanism that checks for new emails every 60 seconds

- Uses Gmail push notifications (Pub/Sub) as an alternative with fallback to polling

- Emits events when new emails arrive using Node.js EventEmitter

- Tracks processed email IDs to avoid duplicates using a Set

- Implements start(), stop(), and getStatus() methods

- Logs each check with timestamp and count of new messages

- Handles errors gracefully and continues running

---

### \*Session 3: Slack Integration (2-3 hours)\*

\*Prompt 8: Slack Connection Setup\*

Create src/integrations/slack/connection.ts that:

- Uses @slack/bolt library to create an App instance

- Loads bot token and signing secret from environment variables

- Implements WebSocket mode (socket mode) for receiving events without public URLs

- Includes connection verification on startup

- Exports getSlackApp() function

- Has detailed error messages for authentication failures

- Includes retry logic for connection issues

\*Prompt 9: Slack Event Listener\*

Create src/integrations/slack/listener.ts that:

- Listens to message events in specific channels (configured via env: SLACK\_MONITORED\_CHANNELS)

- Filters out bot messages to avoid loops

- Parses message data into structured format: { id, user, channel, text, timestamp, thread\_ts, attachments }

- Handles threaded messages and extracts context

- Implements mention detection (when bot is @mentioned)

- Emits events using EventEmitter for new messages

- Includes getUserInfo(userId) to get display names

- Logs all received messages with user and channel info

\*Prompt 10: Slack Action Sender\*

Create src/integrations/slack/actions.ts that:

- Implements sendMessage(channel, text, blocks?) for formatted messages

- Implements sendDirectMessage(userId, text) for DMs

- Implements addReaction(channel, timestamp, emoji) to react to messages

- Implements updateMessage(channel, timestamp, newText) for editing

- Uses Slack Block Kit for rich formatting

- Includes error handling for invalid channels or permissions

- Implements rate limiting (max 1 message per second)

- Logs all sent messages

---

## 📅 DAY 2: Google Services & Composio Integration

### \*Session 4: Google Sheets Integration (2 hours)\*

\*Prompt 11: Google Sheets Reader\*

Create src/integrations/sheets/reader.ts that:

- Uses googleapis library with the same OAuth2 client as Gmail

- Implements readSheet(spreadsheetId, range) that returns 2D array of values

- Implements getSheetMetadata(spreadsheetId) for sheet names and properties

- Implements watchSheet(spreadsheetId, range) that polls for changes every 2 minutes

- Parses data with headers (first row) into array of objects

- Handles empty cells and formatting

- Includes error handling for invalid spreadsheet IDs or permissions

- Emits events when data changes (compares with previous state)

\*Prompt 12: Google Sheets Writer\*

Create src/integrations/sheets/writer.ts that:

- Implements appendRow(spreadsheetId, range, values[]) to add new rows

- Implements updateCell(spreadsheetId, range, value) for single cell updates

- Implements batchUpdate(spreadsheetId, updates[]) for multiple changes

- Validates data types before writing (numbers, strings, dates)

- Includes rollback mechanism if write fails

- Logs all write operations with timestamp

- Handles API quota limits with exponential backoff

---

### \*Session 5: Google Drive Integration (1-2 hours)\*

\*Prompt 13: Drive File Manager\*

Create src/integrations/drive/file-manager.ts that:

- Uses googleapis Drive API v3

- Implements uploadFile(file, folderId, metadata) for file uploads

- Implements createFolder(name, parentId) for folder organization

- Implements moveFile(fileId, newFolderId) to organize files

- Implements listFiles(folderId, query?) to search files

- Implements downloadFile(fileId) to retrieve file content

- Handles file permissions and sharing settings

- Includes progress tracking for large uploads

- Logs all file operations

\*Prompt 14: Drive Email Attachment Handler\*

Create src/integrations/drive/attachment-handler.ts that:

- Accepts email attachment data from Gmail integration

- Automatically uploads attachments to Drive in organized folders by date (e.g., "2025-10/invoices/")

- Extracts file metadata (size, type, sender)

- Implements smart folder creation based on attachment type (invoices, reports, images)

- Returns Drive file links

- Handles duplicate files (appends counter or skips)

- Includes virus scanning check using Drive API

---

### \*Session 6: Composio Tool Router Setup (2-3 hours)\*

\*Prompt 15: Composio Authentication Manager\*

Create src/integrations/composio/auth-manager.ts that:

- Uses Composio SDK from npm

- Implements authenticateApp(appName) for Notion, Trello using Composio's auth flow

- Stores session tokens securely

- Implements refreshAuth(appName) for token renewal

- Validates authentication status before tool execution

- Includes fallback error messages if auth fails

- Exports authenticated clients for each app

- Logs authentication events

\*Prompt 16: Composio Notion Tools\*

Create src/integrations/composio/notion-tools.ts that:

- Uses Composio Tool Router to connect to Notion

- Implements createPage(databaseId, properties) for task creation

- Implements updatePage(pageId, properties) for task updates

- Implements queryDatabase(databaseId, filter, sort) for searching tasks

- Validates properties against Notion database schema

- Handles rich text formatting for descriptions

- Includes error handling for invalid database IDs

- Logs all Notion operations with page IDs

- Returns operation results with success/failure status

\*Prompt 17: Composio Trello Tools\*

Create src/integrations/composio/trello-tools.ts that:

- Uses Composio Tool Router to connect to Trello

- Implements createCard(listId, name, desc, due, labels[]) for task creation

- Implements moveCard(cardId, newListId) for workflow transitions

- Implements addComment(cardId, text) for updates

- Implements getCard(cardId) to fetch card details

- Handles label creation if they don't exist

- Includes attachment support for Drive links

- Logs all Trello operations

- Returns card URLs for easy access

\*Prompt 18: Composio Tool Executor\*

Create src/integrations/composio/executor.ts that:

- Acts as a unified interface for executing tools across Notion and Trello

- Implements executeTool(toolName, params) that routes to correct integration

- Includes pre-execution validation of parameters

- Implements queue system for rate-limited operations

- Has retry logic with exponential backoff (3 attempts)

- Logs execution time and success/failure

- Returns standardized response format: { success: boolean, data: any, error?: string }

- Includes rollback mechanism for failed operations

---

## 📅 DAY 3: Integration Testing & Error Handling

### \*Session 7: Integration Hub & Event Router (2-3 hours)\*

\*Prompt 19: Central Event Hub\*

Create src/integrations/event-hub.ts that:

- Extends Node.js EventEmitter

- Aggregates events from all integrations (Gmail, Slack, Sheets)

- Implements subscribe(eventType, handler) for other modules to listen

- Implements emit(eventType, data) with structured event format: { source, type, timestamp, data, metadata }

- Includes event queue for burst handling

- Logs all events to a central event log file

- Implements getEventHistory(source, limit) for debugging

- Has event filtering by priority or source

\*Prompt 20: Integration Manager\*

Create src/integrations/manager.ts that:

- Initializes all integrations on startup

- Implements startAll() to connect all services in parallel

- Implements stopAll() for graceful shutdown

- Implements healthCheck() that tests each integration's connection

- Returns status dashboard data: { gmail: "connected", slack: "connected", sheets: "connected", etc. }

- Handles individual integration failures without crashing

- Implements reconnection logic for dropped connections

- Logs startup sequence and any failures

---

### \*Session 8: Error Recovery & Resilience (2 hours)\*

\*Prompt 21: Circuit Breaker Pattern\*

Create src/utils/circuit-breaker.ts that:

- Implements circuit breaker pattern for external API calls

- Has three states: CLOSED (normal), OPEN (failing), HALF\_OPEN (testing recovery)

- Opens circuit after 5 consecutive failures

- Closes circuit after 2 successful calls in HALF\_OPEN state

- Implements call(fn, fallback?) method that wraps API calls

- Returns cached data or fallback during OPEN state

- Logs state transitions with reasons

- Exports createCircuitBreaker(name, options) factory function

\*Prompt 22: Retry Queue for Failed Operations\*

Create src/utils/retry-queue.ts that:

- Implements a persistent queue for failed operations using JSON file storage

- Stores failed operation details: { id, type, params, attempts, lastError, timestamp }

- Implements enqueue(operation) to add failed operations

- Implements processQueue() that retries operations with exponential backoff

- Limits retry attempts to 5 with increasing delays (1min, 5min, 15min, 1hr, 6hr)

- Removes operations after max attempts and logs to failed-operations.log

- Runs automatically on startup and every 5 minutes

- Returns retry statistics

\*Prompt 23: Health Check Endpoint\*

Create src/utils/health-check.ts that:

- Implements comprehensive health checks for all integrations

- Tests each service connection (ping Gmail, Slack, Sheets, etc.)

- Checks disk space in logs directory

- Checks memory usage

- Returns health status: { status: "healthy" | "degraded" | "unhealthy", services: {...}, timestamp }

- Implements runHealthCheck() that runs all checks in parallel with timeout

- Logs health status every 10 minutes

- Exports getHealthStatus() for dashboard display

---

## 📅 DAY 4: Testing, Documentation & Polish

### \*Session 9: Integration Testing Suite (2-3 hours)\*

\*Prompt 24: Test Gmail Integration\*

Create tests/integrations/gmail.test.ts that:

- Tests connection with valid credentials

- Tests fetching unread messages (mocked response)

- Tests email parsing with multipart MIME

- Tests error handling for invalid credentials

- Tests listener polling mechanism

- Tests duplicate detection

- Uses Jest or Mocha with chai

- Includes setup and teardown for clean state

- Logs test results

\*Prompt 25: Test Slack Integration\*

Create tests/integrations/slack.test.ts that:

- Tests connection with valid tokens

- Tests message listener with mocked events

- Tests message sending with different formats

- Tests error handling for invalid channels

- Tests rate limiting behavior

- Tests thread handling

- Mocks Slack API responses

- Includes comprehensive assertions

\*Prompt 26: Test Composio Tools\*

Create tests/integrations/composio.test.ts that:

- Tests Notion page creation with various property types

- Tests Trello card creation with labels and due dates

- Tests error handling for invalid parameters

- Tests retry logic on failures

- Tests queue system for rate-limited operations

- Mocks Composio API responses

- Validates response formats

\*Prompt 27: End-to-End Integration Test\*

Create tests/e2e/full-pipeline.test.ts that:

- Simulates: Gmail receives email → EventHub emits event → (to be processed by Member 2)

- Tests event flow through all integrations

- Tests error propagation and recovery

- Uses test environment with mocked external APIs

- Validates data transformations at each step

- Measures end-to-end latency

- Logs complete execution trace

---

### \*Session 10: Documentation & Setup Guide (2-3 hours)\*

\*Prompt 28: Comprehensive README\*

Create README.md with the following sections:

1. Project Overview - what the system does in 2-3 sentences

2. Architecture Diagram - ASCII diagram showing data flow: Gmail/Slack/Sheets → EventHub → LLM → Actions → Notion/Trello/Drive

3. Prerequisites - Node.js 18+, npm, API keys needed

4. Installation Steps:

- Clone repository

- npm install

- Copy .env.example to .env

- Fill in API credentials (with links to getting keys)

5. Authentication Setup:

- Gmail OAuth setup (step-by-step with screenshots)

- Slack app creation and token generation

- Google Sheets/Drive API enabling

- Composio account setup

6. Running the Application:

- npm run dev for development

- npm start for production

- npm test for running tests

7. Project Structure - explanation of each directory

8. Troubleshooting - common errors and solutions

9. Contributing guidelines

\*Prompt 29: API Authentication Guide\*

Create docs/AUTHENTICATION.md that:

- Has step-by-step guide for Gmail OAuth:

\* Creating Google Cloud project

\* Enabling Gmail API

\* Creating OAuth credentials

\* Getting client ID and secret

\* Running first-time auth flow

- Has step-by-step guide for Slack:

\* Creating Slack app

\* Adding bot scopes (channels:read, chat:write, etc.)

\* Installing app to workspace

\* Getting bot token and signing secret

- Has guide for Google Sheets/Drive API enabling

- Has guide for Composio authentication

- Includes screenshots and troubleshooting tips

- Explains what each permission/scope does

\*Prompt 30: Integration Architecture Documentation\*

Create docs/ARCHITECTURE.md that:

- Explains the integration layer architecture

- Describes BaseIntegration abstract class pattern

- Documents event flow through EventHub

- Explains error handling and retry strategies

- Describes rate limiting implementation

- Documents circuit breaker pattern usage

- Includes sequence diagrams for key operations

- Lists all exported functions from each integration module

- Explains how other team members should interface with your modules

\*Prompt 31: Troubleshooting Guide\*

Create docs/TROUBLESHOOTING.md that:

- Lists common errors and solutions:

\* "Gmail authentication failed" → check credentials, re-run OAuth

\* "Slack connection timeout" → check internet, verify tokens

\* "Rate limit exceeded" → explains backoff strategy

\* "Integration health check failed" → how to diagnose

- Includes debugging tips:

\* How to enable debug logging

\* Where to find log files

\* How to test individual integrations

- Has FAQ section:

\* How often does the system poll?

\* How to add new channels to monitor?

\* How to reset authentication?

- Emergency procedures:

\* How to stop all integrations

\* How to clear retry queue

\* How to reset to fresh state

- Logs metrics every 5 minutes

## 📋 Deliverables Checklist for Member 1

### Code Deliverables

- [ ] Complete integration modules for Gmail, Slack, Google Sheets, Drive

- [ ] Composio Tool Router setup for Notion and Trello

- [ ] Event Hub for aggregating signals

- [ ] Integration Manager for lifecycle management

- [ ] Circuit breaker and retry queue for resilience

- [ ] Health check system

- [ ] Comprehensive test suite

### Documentation Deliverables

- [ ] README.md with complete setup instructions

- [ ] AUTHENTICATION.md with step-by-step auth guides

- [ ] ARCHITECTURE.md explaining integration design

- [ ] TROUBLESHOOTING.md with common issues

- [ ] .env.example with all required variables

- [ ] Inline code comments (JSDoc) for all public functions

### Testing Deliverables

- [ ] Unit tests for each integration

- [ ] Integration tests for cross-module communication

- [ ] End-to-end test simulating full pipeline

- [ ] Demo mode with sample data

### Handoff to Team

- [ ] Integration APIs documented for Member 2 (how to subscribe to events)

- [ ] Action execution APIs documented for Member 3 (how to trigger actions)

- [ ] Status data format documented for Member 4 (dashboard metrics)

- [ ] All modules export clear TypeScript interfaces

---

## 🎯 Success Criteria

1. \*All integrations connect successfully\* - No authentication errors

2. \*Real-time monitoring works\* - Events trigger within 60 seconds

3. \*Error handling is robust\* - System continues running despite individual failures

4. \*Documentation is complete\* - New developer can set up in 30 minutes

5. \*Tests pass\* - 90%+ code coverage on integration modules

6. \*Demo works smoothly\* - Can showcase without real API calls

---

## 🚨 Critical Reminders

- \*Never commit .env files\* - Always use .env.example

- \*Test with fresh npm install\* - Ensure dependencies are correct

- \*Log everything\* - Helps debug issues for the team

- \*Use TypeScript strictly\* - Catch errors at compile time

- \*Rate limit proactively\* - Don't hit API quotas

- \*Fail gracefully\* - One broken integration shouldn't crash the system

This is your complete roadmap. Use GitHub Copilot with these prompts in sequence, and you'll build a rock-solid integration layer that the entire team depends on!