

# Satellite Application Specification

---

**Version:** 1.0

**Date:** November 3, 2025

**Purpose:** Requirements for satellite apps in the SFG Aluminium Ecosystem

## Overview

---

Satellite applications are autonomous apps that register with and receive orchestration from Nexus (the central conductor).

## Core Requirements

---

### 1. Self-Registration

Every satellite app must be able to:

- Clone the central repository
- Create registration JSON file
- Backup its source code
- Extract business logic as JSON
- Push to GitHub
- Create notification issue

### 2. MCP Server (Optional but Recommended)

Implement MCP server to expose:

- **Tools:** Actions Nexus can invoke
- **Resources:** Data Nexus can access
- **Prompts:** Templates for common tasks

### 3. Webhook Receiver

Implement webhook endpoint to receive:

- New instructions from Nexus
- Configuration updates
- Orchestration commands

### 4. Health Monitoring

Expose health check endpoint:

- `GET /api/health` - Returns app status
- Response: `{ "status": "healthy", "version": "1.0.0", "uptime": 12345 }`

## Registration Format

### app-registration.json

```
{  
  "app_id": "unique-app-id",  
  "app_name": "Full Application Name",  
  "app_url": "https://app.abacusai.app",  
  "app_category": "core|support|experimental",  
  "version": "1.0.0",  
  "registered_at": "2025-11-03T00:00:00Z",  
  "technical": {  
    "technology_stack": "Next.js 14, TypeScript, PostgreSQL",  
    "database": "PostgreSQL 15",  
    "hosting": "Abacus.AI",  
    "mcp_enabled": true,  
    "mcp_endpoint": "https://app.abacusai.app/mcp",  
    "api_endpoints": ["/api/health", "/api/data"]  
  },  
  "business": {  
    "primary_function": "What this app does",  
    "target_users": "Who uses it",  
    "key_features": ["Feature 1", "Feature 2"],  
    "staff_replaced": 2,  
    "annual_savings_gbp": 50000  
  },  
  "integration": {  
    "integrates_with": ["app-001", "app-002"],  
    "data_sources": ["Database", "API"],  
    "data_outputs": ["Reports", "Notifications"]  
  },  
  "status": "registered",  
  "health": "healthy"  
}
```

## business-logic.json

```
{  
    "app_id": "unique-app-id",  
    "extracted_at": "2025-11-03T00:00:00Z",  
    "business_rules": [  
        {  
            "rule_id": "rule_001",  
            "name": "Rule Name",  
            "description": "What it does",  
            "condition": "When it applies",  
            "action": "what happens",  
            "priority": "high|medium|low"  
        }  
    ],  
    "workflows": [  
        {  
            "workflow_id": "workflow_001",  
            "name": "Workflow Name",  
            "steps": [  
                {  
                    "step": 1,  
                    "action": "Action description",  
                    "responsible": "System/User",  
                    "duration_minutes": 5  
                }  
            ]  
        }  
    ],  
    "calculations": [],  
    "validations": [],  
    "integrations": []  
}
```

# MCP Server Implementation

## Basic Structure

```
// /api/mcp/route.ts
import { NextRequest, NextResponse } from 'next/server';

export async function POST(request: NextRequest) {
  const { method, params } = await request.json();

  switch (method) {
    case 'tools/list':
      return NextResponse.json({
        tools: [
          {
            name: 'get_data',
            description: 'Retrieve data from this app',
            inputSchema: {
              type: 'object',
              properties: {
                query: { type: 'string' }
              }
            }
          }
        ]
      });
    case 'tools/call':
      const { name, arguments: args } = params;
      if (name === 'get_data') {
        const data = await getData(args.query);
        return NextResponse.json({ content: [{ type: 'text', text: JSON.stringify(data) }] });
      }
      break;

    case 'resources/list':
      return NextResponse.json({
        resources: [
          {
            uri: 'app://data',
            name: 'Application Data',
            mimeType: 'application/json'
          }
        ]
      });
    default:
      return NextResponse.json({ error: 'Unknown method' }, { status: 400 });
  }
}
```

# Webhook Implementation

## Webhook Receiver

```
// /api/github-webhook/route.ts
import { NextRequest, NextResponse } from 'next/server';
import crypto from 'crypto';

export async function POST(request: NextRequest) {
    // Verify signature
    const signature = request.headers.get('x-hub-signature-256');
    const body = await request.text();

    const secret = process.env.GITHUB_WEBHOOK_SECRET;
    const hmac = crypto.createHmac('sha256', secret);
    const digest = `sha256=${hmac.update(body).digest('hex')}`;

    if (signature !== digest) {
        return NextResponse.json({ error: 'Invalid signature' }, { status: 401 });
    }

    // Process webhook
    const payload = JSON.parse(body);

    if (payload.commits) {
        for (const commit of payload.commits) {
            const hasInstructions = commit.added.some((file: string) =>
                file.startsWith('instructions/satellites/') &&
                file.includes(process.env.APP_ID)
            );

            if (hasInstructions) {
                await pullAndImplementInstructions();
                await createImplementationReport(commit);
            }
        }
    }
}

return NextResponse.json({ status: 'processed' });
}
```

# Behavioral Requirements

## Autonomous Operation

- Monitor for new instructions via webhooks
- Pull and implement instructions automatically
- Report completion via GitHub issues
- Handle errors gracefully

## Communication

- Respond to MCP calls from Nexus
- Expose health status
- Report errors and issues

- Provide progress updates

## Data Management

- Backup code regularly
- Version business logic
- Track changes
- Maintain data integrity

# Performance Requirements

---

## Response Time

- Health checks: <100ms
- MCP calls: <1s
- Webhook processing: <5s

## Availability

- Uptime: 99%+
- Recovery time: <10 minutes

## Scalability

- Handle 100+ MCP calls/day
- Process 10+ instructions/day

# Success Metrics

---

## Registration

- Complete registration within 30 minutes
- All required files present
- Valid JSON format

## Integration

- MCP server operational
- Webhook receiver working
- Health checks passing

## Orchestration

- 95%+ instruction completion rate
- <1 hour average implementation time
- Automatic reporting

# Support

---

## Questions

Create GitHub issue with label “question”

## Errors

Create GitHub issue with label “error”

## Registration Help

See [/instructions/satellites/self-register.md](#)