CONSTITUTION AI AGENT - PRODUCT REQUIREMENTS DOCUMENT (PRD)

Version: 1.0
Date: January 29, 2026
Owner: Ayush
Status: Planning Phase

TABLE OF CONTENTS

1. Executive Summary
2. Goals & Success Metrics
3. Technical Architecture
4. Feature Specifications
5. Data Models
6. API Specifications
7. Implementation Phases
8. Testing Strategy
9. Deployment Strategy
10. Monitoring & Maintenance
11. Cost Analysis
12. Risk Mitigation

---

1. EXECUTIVE SUMMARY

1.1 Problem Statement

User needs an AI-powered accountability system to enforce life constitution across multiple domains (physical health, career, mental health, relationships, wealth). Current manual tracking is unsustainable and prone to failure during crisis periods.

1.2 Solution Overview

Multi-agent AI system built on LangGraph + Vertex AI (Gemini 2.0 Flash) that:

- Conducts daily check-ins (9 PM IST)

- Detects behavior patterns and violations
- Generates weekly/monthly analytical reports
- Provides CBT-style emotional processing
- Executes 3-level interventions (nudge/warning/veto)
- Monitors for crisis (ghosting detection)

1.3 Key Constraints

- **Budget:** $5/month maximum
- **Platform:** GCP exclusively (Vertex AI, Cloud Run, Firestore)
- **Framework:** LangChain/LangGraph
- **Language:** Python 3.11+
- **Interface:** Telegram Bot (free, no message costs)

1.4 Success Criteria

- ✅ 90%+ daily check-in completion rate
- ✅ Response time <5 seconds for check-ins
- ✅ Pattern detection accuracy >85%
- ✅ Weekly/monthly reports generated on-time 100%
- ✅ Monthly cost <$5
- ✅ System uptime >99%

---

2. GOALS & SUCCESS METRICS

2.1 Primary Goals

Goal 1: Daily Accountability

- **Metric:** Check-in completion rate >90%
- **Target:** 27/30 days per month minimum
- **Tracking:** Firestore timestamp logs

Goal 2: Pattern Detection

- **Metric:** Detect constitution violations within 24 hours
- **Target:** 100% of critical violations caught (sleep <6hrs 3+ days, missed workouts 3+ days)
- **Tracking:** Pattern detection agent logs

Goal 3: Intervention Effectiveness

- **Metric:** User behavior correction after intervention

- **Target:** >70% of interventions lead to behavior change within 48 hours
- **Tracking:** Pre/post intervention metrics comparison

Goal 4: Report Quality

- **Metric:** User satisfaction with weekly/monthly reports
- **Target:** Reports generated on-time 100%, contain actionable insights
- **Tracking:** Manual review + feedback

Goal 5: Cost Efficiency

- **Metric:** Monthly operational cost
- **Target:** <$5/month all-inclusive
- **Tracking:** GCP billing dashboard

2.2 Secondary Goals

Goal 6: System Reliability

- **Metric:** Uptime percentage
- **Target:** >99% (max 7 hours downtime/month)
- **Tracking:** Cloud Monitoring

Goal 7: Response Performance

- **Metric:** Check-in response latency
- **Target:** <5 seconds 95th percentile
- **Tracking:** Cloud Trace

---

3. TECHNICAL ARCHITECTURE

3.1 System Architecture Diagram

```
┌──────────────────────────────────────────────────────────────────
──┐
│                    USER LAYER                    │
│                 Telegram Mobile App                 │
└────────────────────────────────┬─────────────────────────────────
──┘
                    │
                    ▼
┌──────────────────────────────────────────────────────────────────
──┐
```

```
│             INTERFACE LAYER              │
│  ┌─────────────────────────────────────────────────────────┐  │
│  │ Telegram Bot API (python-telegram-bot)      │  │
│  │ - Webhook receiver                    │  │
│  │ - Message parser                     │  │
│  │ - Response formatter                  │  │
│  └─────────────────────────────────────────────────────────┘  │
└──────────────────────────────┬───────────────────────────────
──┘
              │
              ▼
┌─────────────────────────────────────────────────────────────
──┐
│             APPLICATION LAYER              │
│  ┌─────────────────────────────────────────────────────────┐  │
│  │    FastAPI Application (Cloud Run)         │  │
│  │ ┌────────────────────────────────────────────────┐ │  │
│  │ │ LangGraph Supervisor Agent         │ │  │
│  │ │ - Intent classification          │ │  │
│  │ │ - Agent routing                │ │  │
│  │ │ - State management              │ │  │
│  │ └───────────────────┬────────────────────────────┘ │  │
│  │         │            │  │
│  │ ┌───────────────────┴────────────────────────────┐ │  │
│  │ │    Sub-Agent Ecosystem           │ │  │
│  │ │ ┌─────────────────────────────────────────┐ │ │  │
│  │ │ │ 1. CheckInAgent              │ │ │  │
│  │ │ │   - Question flow            │ │ │  │
│  │ │ │   - Response validation         │ │ │  │
│  │ │ │   - Compliance calculation       │ │ │  │
│  │ │ └─────────────────────────────────────────┘ │ │  │
│  │ │ ┌─────────────────────────────────────────┐ │ │  │
│  │ │ │ 2. PatternDetectionAgent         │ │ │  │
│  │ │ │   - Time series analysis        │ │ │  │
│  │ │ │   - Rule-based detection        │ │ │  │
│  │ │ │   - Anomaly detection          │ │ │  │
│  │ │ └─────────────────────────────────────────┘ │ │  │
│  │ │ ┌─────────────────────────────────────────┐ │ │  │
│  │ │ │ 3. ReportAgent              │ │ │  │
│  │ │ │   - Data aggregation          │ │ │  │
│  │ │ │   - Graph generation          │ │ │  │
│  │ │ │   - LLM summarization          │ │ │  │
│  │ │ └─────────────────────────────────────────┘ │ │  │
│  │ │ ┌─────────────────────────────────────────┐ │ │  │
│  │  │  │  4. EmotionalProcessingAgent            │  │  │  │
│  │  │  │     - CBT reframing                │  │  │  │
│  │  │  │     - Sentiment analysis                │  │  │  │
│  │  │  │     - Coping strategy suggestions        │  │  │  │
│  │  │  └──────────────────────────────────────────┘  │  │  │
│  │  │  ┌──────────────────────────────────────────┐  │  │  │
│  │  │  │  5. InterventionAgent                │  │  │  │
│  │  │  │     - Severity assessment              │  │  │  │
│  │  │  │     - Message generation              │  │  │  │
│  │  │  │     - Escalation logic              │  │  │  │
│  │  │  └──────────────────────────────────────────┘  │  │  │
│  │  └──────────────────────────────────────────────────┘  │  │
│  └──────────────────────────────────────────────────────────┘  │
└───────────────────────────┬───────────────────────────────────────
──┘
                            │
                            ▼
┌───────────────────────────────────────────────────────────────────
──┐
│                 AI/LLM LAYER                         │
│  ┌──────────────────────────────────────────────────────────┐  │
│  │  Vertex AI - Gemini 2.0 Flash                     │  │
│  │  - Intent classification                        │  │
│  │  - Response generation                          │  │
│  │  - Pattern summarization                        │  │
│  │  - Emotional processing                         │  │
│  └──────────────────────────────────────────────────────────┘  │
└───────────────────────────┬───────────────────────────────────────
──┘
                            │
                            ▼
┌───────────────────────────────────────────────────────────────────
──┐
│                DATA LAYER                           │
│  ┌──────────────────────────────────────────────────────────┐  │
│  │  Firestore (Primary Database)                     │  │
│  │  Collections:                                 │  │
│  │  - users/                                    │  │
│  │  - daily_checkins/{user_id}/checkins/              │  │
│  │  - metrics/{user_id}/weekly/                    │  │
│  │  - patterns/{user_id}/detected/                 │  │
│  │  - interventions/{user_id}/history/             │  │
│  │  - emotional_sessions/{user_id}/sessions/           │  │
│  └──────────────────────────────────────────────────────────┘  │
│  ┌──────────────────────────────────────────────────────────┐  │
│  │ Cloud Storage (Media/Exports)                            │  │
│  │ - Generated graphs (PNG)                                 │  │
│  │ - Report PDFs                                            │  │
│  │ - Export CSVs                                            │  │
│  └──────────────────────────────────────────────────────────┘  │
└───────────────────────────┬────────────────────────────────────┘
                            │
                            ▼
┌────────────────────────────────────────────────────────────────┐
│                    SCHEDULING LAYER                            │
│  ┌──────────────────────────────────────────────────────────┐  │
│  │ Cloud Scheduler                                          │  │
│  │ - Daily check-in trigger (9 PM IST)                      │  │
│  │ - Weekly report trigger (Sunday 9 AM IST)                │  │
│  │ - Monthly review trigger (1st Sunday 9 AM IST)           │  │
│  │ - Pattern detection scan (every 6 hours)                 │  │
│  │ - Ghosting check (daily 10 PM IST)                       │  │
│  └──────────────────────────────────────────────────────────┘  │
└────────────────────────────────────────────────────────────────┘
                            │
                            ▼
┌────────────────────────────────────────────────────────────────┐
│                  DASHBOARD LAYER (Optional)                    │
│  ┌──────────────────────────────────────────────────────────┐  │
│  │ Streamlit Dashboard (Cloud Run)                          │  │
│  │ - Real-time metrics                                      │  │
│  │ - Interactive graphs (Plotly)                            │  │
│  │ - Pattern visualization                                  │  │
│  │ - Export functionality                                   │  │
│  └──────────────────────────────────────────────────────────┘  │
└────────────────────────────────────────────────────────────────┘
```

3.2 Technology Stack

Layer	Technology	Version	Purpose
Runtime	Python	3.11+	Primary language

Framework	FastAPI	0.109+	Web application
Bot Framework	python-telegram-bot	21.0+	Telegram integration
AI Orchestration	LangGraph	0.0.40+	Multi-agent system
AI Framework	LangChain	0.1.0+	LLM tooling
LLM	Vertex AI Gemini 2.0 Flash	Latest	Language model
Database	Firestore	N/A	NoSQL storage
Storage	Cloud Storage	N/A	Media files
Compute	Cloud Run	N/A	Container hosting
Scheduling	Cloud Scheduler	N/A	Cron jobs
Visualization	Plotly	5.18+	Graph generation
Dashboard	Streamlit	1.30+	Web dashboard
Data Processing	Pandas	2.2+	Data manipulation
HTTP Client	httpx	0.26+	Async requests
Validation	Pydantic	2.5+	Data validation
Testing	pytest	8.0+	Unit testing
Monitoring	Cloud Logging	N/A	Logging
Metrics	Cloud Monitoring	N/A	Observability

3.3 Infrastructure Components

Cloud Run Configuration:

Service: constitution-agent
Region: asia-south1 (Mumbai - closest to Hyderabad)
CPU: 1
Memory: 512Mi
Min Instances: 0 (scale to zero)
Max Instances: 3
Concurrency: 80
Timeout: 300s (5 min for report generation)

Firestore Configuration:

```
Location: asia-south1
Mode: Native
Rules: Authenticated access only
Indexes:
  - Collection: daily_checkins, Fields: [user_id, date]
  - Collection: patterns, Fields: [user_id, detected_at, severity]
```

Cloud Scheduler Jobs:

```
Jobs:
  1. daily-checkin-trigger (0 21 * * *, Asia/Kolkata)
  2. weekly-report-trigger (0 9 * * 0, Asia/Kolkata)
  3. monthly-review-trigger (0 9 1-7 * 0, Asia/Kolkata)
  4. pattern-scan-trigger (0 */6 * * *, Asia/Kolkata)
  5. ghosting-check-trigger (0 22 * * *, Asia/Kolkata)
```

---

4. FEATURE SPECIFICATIONS

4.1 FEATURE: Daily Check-In System

Priority: P0 (Critical)
Status: Not Started
Owner: TBD
Effort: 3 days

Description:

Automated daily check-in flow triggered at 9 PM IST that asks 4 constitution-defined questions, validates responses, calculates compliance score, and provides immediate feedback.

User Flow:
1. 9:00 PM IST - Bot sends: "Time for your daily check-in! Ready?"
2. User: "Yes"
3. Bot: "Question 1/5: Constitution Compliance - Did you complete Tier 1 non-negotiables?"
   - Sleep: 7+ hours last night? (Y/N)
   - Training: Completed workout OR scheduled rest day? (Y/N)
   - Deep Work: 2+ hours focused work/study? (Y/N)
   - Zero Porn: No porn today? (Y/N)
   - Boundaries: No toxic interactions/sacrifices? (Y/N)

4. User: "Sleep: Y, Training: Y, Deep Work: Y, Zero Porn: Y, Boundaries: Y"
5. Bot: "Question 2/5: What challenges did you face today? How did you handle them?"
6. User: [Free text response]
7. Bot: "Question 3/5: Rate today 1-10 on constitution alignment. Why that score?"
8. User: "8 - missed one study hour but overall solid"
9. Bot: "Question 4/5: What's tomorrow's #1 priority and biggest potential obstacle?"
10. User: "Priority: Complete system design chapter. Obstacle: Might get late meeting."
11. Bot: [Generates response with streak, patterns, feedback]

Technical Requirements:

Input Validation:

- Question 1: Parse Y/N responses (flexible: yes/no/y/n/1/0)
- Question 2: Min 10 chars, max 500 chars
- Question 3: Rating 1-10 (int) + reason (min 10 chars)
- Question 4: Priority (min 10 chars) + Obstacle (min 10 chars)

Compliance Score Calculation:

```
def calculate_compliance_score(responses: dict) -> float:
    """
    Tier 1 non-negotiables: 5 items (sleep, training, deep_work, zero_porn, boundaries)
    Each Y = 1 point, N = 0 point
    Score = (points / 5) * 100
    """
    tier1 = responses.get('tier1_non_negotiables', {})
    points = sum([
        tier1.get('sleep', False),
        tier1.get('training', False),
        tier1.get('deep_work', False),
        tier1.get('zero_porn', False),
        tier1.get('boundaries', False)
    ])
    return (points / 5) * 100
```

Response Generation: LLM prompt to generate feedback based on:

- Compliance score
- Historical streak
- Detected patterns (if any)
- Constitution mode (optimization/maintenance/survival)

Acceptance Criteria:

- [ ] Bot sends check-in prompt at 9 PM IST daily
- [ ] User can complete check-in in <3 minutes
- [ ] All 4 questions answered and stored in Firestore
- [ ] Compliance score calculated correctly
- [ ] Streak counter updates accurately
- [ ] Feedback response generated in <5 seconds
- [ ] Check-in can be triggered manually with `/checkin` command
- [ ] Missed check-ins logged (for ghosting detection)

Implementation Checklist:

Backend:

- [ ] Create `CheckInAgent` class in `agents/checkin_agent.py`
- [ ] Implement question flow state machine
- [ ] Add response validation functions
- [ ] Implement compliance score calculator
- [ ] Create Firestore write function for check-in data
- [ ] Add streak counter logic
- [ ] Implement LLM feedback generation
- [ ] Add error handling for incomplete check-ins

Bot Integration:

- [ ] Create `/checkin` command handler
- [ ] Implement conversational state management
- [ ] Add timeout handler (15 min inactivity)
- [ ] Create "resume check-in" functionality
- [ ] Add inline keyboard for Y/N responses

Scheduling:

- [ ] Create Cloud Scheduler job for 9 PM trigger
- [ ] Implement scheduled trigger endpoint
- [ ] Add reminder logic (if no response in 30 min, send reminder)

Testing:

- [ ] Unit tests for compliance score calculation
- [ ] Unit tests for response validation
- [ ] Integration test for full check-in flow
- [ ] Test timezone handling (IST)
- [ ] Test state persistence across messages

4.2 FEATURE: Pattern Detection System

Priority: P0 (Critical)
Status: Not Started
Owner: TBD
Effort: 4 days

Description:

Automated system that analyzes historical check-in data to detect constitution violations, behavioral patterns, and trigger appropriate interventions.

Pattern Types to Detect:

Critical Patterns (Immediate Intervention):

1. **Sleep Degradation:** <6 hours sleep for 3+ consecutive nights
2. **Training Abandonment:** Missed workouts 3+ consecutive days (not rest days)
3. **Porn Relapse:** 3+ instances in one week
4. **Boundary Violations:** 3+ instances in one week
5. **Check-in Ghosting:** 3+ consecutive missed check-ins

Warning Patterns (Monitor & Warn):

1. **Sleep Reduction:** <7 hours sleep for 2+ nights
2. **Training Skip:** Missed 2 workouts consecutively
3. **Productivity Drop:** <2 hours deep work for 2+ days
4. **Compliance Decline:** Score <70% for 3+ days

Positive Patterns (Reinforce):

1. **Streak Milestones:** 7, 14, 30, 60, 90, 180, 365 days
2. **Perfect Weeks:** 100% compliance for 7 days
3. **Recovery Success:** Bounced back after violation within 48 hours

Detection Logic:

```
class PatternDetectionAgent:

    PATTERN_RULES = {
        'sleep_degradation': {
            'condition': lambda data: (
                len([d for d in data[-3:] if d['sleep_hours'] < 6]) >= 3
            ),
            'severity': 'warning',
            'message_template': "You've violated sleep protocol {count} nights in a row. Constitution says: sleep is non-negotiable. What's blocking you?"
        },
        'training_abandonment': {
            'condition': lambda data: (
                len([d for d in data[-3:] if not d['workout_completed'] and not d['is_rest_day']]) >= 3
            ),
            'severity': 'warning',
            'message_template': "You've missed workout {count} days in a row. This historically leads to spiral. What's happening?"
        },
        # ... more patterns
    }

    def detect_patterns(self, user_id: str) -> List[Pattern]:
        """Scan recent check-ins for pattern matches"""
        recent_data = self.get_recent_checkins(user_id, days=7)
        detected = []

        for pattern_name, rule in self.PATTERN_RULES.items():
            if rule['condition'](recent_data):
                detected.append({
                    'pattern_name': pattern_name,
                    'severity': rule['severity'],
                    'message': self.generate_message(pattern_name, recent_data)
                })

        return detected
```

Intervention Levels:

Level 1: Nudge (No blocking)

- Gentle reminder
- Asks what's wrong
- Suggests solutions
- Example: "Sleep was <7hrs for 2 nights. Constitution recommends earlier bedtime. Want to set a goal for tonight?"

Level 2: Warning (Strong recommendation)

- Firm message
- References constitution explicitly
- Flags historical pattern
- Example: "Sleep <6hrs for 3+ nights. Constitution violation. You're entering spiral pattern that led to 6-month regression in Feb 2025. We need to fix this TODAY."

Level 3: Veto (Strong recommendation + override protocol)

- Blocks or strongly recommends against action
- Requires 500-word override justification
- Escalates to accountability partner (future feature)
- Example: "You want to return to 6x/week training but it's only 4 weeks post-surgery. Constitution graduation criteria NOT met. Veto. Stay in Maintenance Mode."

Acceptance Criteria:

- [ ] All 9 critical + warning patterns detected correctly
- [ ] Detection runs automatically every 6 hours
- [ ] Patterns stored in Firestore with timestamps
- [ ] Interventions triggered based on severity
- [ ] False positive rate <10%
- [ ] Detection latency <2 seconds

Implementation Checklist:

Backend:

- [ ] Create `PatternDetectionAgent` class
- [ ] Implement pattern rule engine
- [ ] Add time-series analysis functions
- [ ] Create pattern storage schema in Firestore
- [ ] Implement severity assessment logic
- [ ] Add historical pattern lookup (for "you did X before" messages)
- [ ] Create pattern resolution tracking

Detection Rules:

- [ ] Implement sleep degradation detection
- [ ] Implement training abandonment detection
- [ ] Implement porn relapse detection
- [ ] Implement boundary violation detection
- [ ] Implement ghosting detection
- [ ] Implement compliance decline detection
- [ ] Implement streak milestone detection

Intervention System:

- [ ] Create `InterventionAgent` class
- [ ] Implement 3-level intervention logic
- [ ] Add message template system
- [ ] Create intervention history tracking
- [ ] Add effectiveness tracking (did user correct behavior?)

Scheduling:

- [ ] Create Cloud Scheduler job for 6-hour scans
- [ ] Implement scan trigger endpoint
- [ ] Add on-demand pattern check after each check-in

Testing:

- [ ] Unit tests for each pattern rule
- [ ] Integration test for full detection flow
- [ ] Test false positive scenarios
- [ ] Test intervention level escalation
- [ ] Test pattern resolution logic

4.3 FEATURE: Weekly Report Generation

Priority: P1 (High)
Status: Not Started
Owner: TBD
Effort: 4 days

Description:

Automated weekly report generated every Sunday at 9 AM IST that aggregates 7 days of data, generates visualizations, detects trends, and provides LLM-generated insights.

Report Structure:

Section 1: Executive Summary

- Overall compliance score (% of week)
- Streak status (days)
- Trend indicator (↑↓→ vs last week)
- Key highlights (wins/losses)

Section 2: Domain-by-Domain Analysis

Physical Health:
- Workouts: 5/6 completed (83%) [Target: 6/6]
- Sleep: Avg 7.2hrs [Target: 7+hrs]
- Protein: Avg 165g [Target: 180g]
- Status: ⚠️ On track but below protein target

Career Development:

- LeetCode: 12 problems solved [Target: 14]
- Deep Work: Avg 2.3hrs/day [Target: 2+hrs]
- Job Applications: 3 sent [Target: 5]
- Status: ⚠️ Behind on applications

Mental Health:
- Check-ins: 7/7 completed (100%)
- Emotional sessions: 2
- Mood avg: 7.5/10
- Status: ✅ Excellent consistency

Wealth:
- SIP deposited: ✅ ₹30K
- Budget adherence: 95%
- Status: ✅ On track

Section 3: Longitudinal Trends (4-week view)

- Sleep hours graph (line chart)
- Workout frequency graph (bar chart)
- Compliance score trend (line chart)
- Mood ratings trend (line chart)

Section 4: Pattern Flags

⚠️ Active Warnings:
- Sleep reduction pattern detected (2 nights <7hrs)
- LeetCode consistency dropped 20% vs last week

✅ Positive Patterns:
- Zero porn for 14 days straight (longest streak in 6 months!)
- 100% check-in completion this week

Section 5: Next Week Priorities

Focus Areas:
1. Career: Increase job applications to 5/week
2. Physical: Hit protein target 180g daily
3. Mental: Continue perfect check-in streak

Suggested Actions:
- Set calendar reminder for job applications (Wed/Fri)
- Prep protein shakes in advance

- Celebrate 14-day milestone 🎉

Graph Specifications:

Graph 1: Sleep Trends (Plotly Line Chart)

```
# X-axis: Last 28 days
# Y-axis: Sleep hours (0-10)
# Horizontal line at 7 hrs (target)
# Color: Green if ≥7, Yellow if 6-7, Red if <6
```

Graph 2: Workout Frequency (Plotly Bar Chart)

```
# X-axis: Last 4 weeks
# Y-axis: Workouts completed (0-6)
# Target line at 6
# Color gradient based on % of target
```

Graph 3: Compliance Score (Plotly Line Chart)

```
# X-axis: Last 28 days
# Y-axis: Daily compliance % (0-100)
# Moving average line (7-day)
# Color zones: Green >80%, Yellow 60-80%, Red <60%
```

Graph 4: Domain Radar Chart (Plotly Polar)

```
# Axes: Physical, Career, Mental, Wealth, Relationships
# Current week: Solid line
# Last week: Dotted line (comparison)
# Scale: 0-100% compliance
```

LLM Report Generation:

```
def generate_weekly_summary(data: dict) -> str:
    """
    LLM prompt to generate narrative summary

    Inputs:
    - Compliance scores (overall, by domain)
    - Trend data (vs last week)
    - Detected patterns
    - User's current mode (optimization/maintenance/survival)
    - Constitution context

    Output:
    - 200-300 word summary in Ayush's constitution tone
    - Supportive but firm
    - Evidence-based observations
    - Actionable next steps
    """
    prompt = f"""
    You are Ayush's constitution AI. Generate a weekly summary report.

    Data:
    - Overall compliance: {data['compliance_overall']}%
    - Physical: {data['physical_score']}%
    - Career: {data['career_score']}%
    - Sleep avg: {data['sleep_avg']} hrs
    - Workouts: {data['workouts_completed']}/6
    - Patterns detected: {data['patterns']}

    Constitution mode: {data['mode']}

    Guidelines:
    - Be supportive but firm (coach tone)
    - Reference constitution when relevant
    - Celebrate wins, address losses directly
    - Provide 2-3 specific actions for next week
    - Keep it 200-300 words
    """
```

Acceptance Criteria:

- [ ] Report generated every Sunday 9 AM IST
- [ ] All 5 sections populated correctly
- [ ] 4 graphs generated and uploaded to Cloud Storage
- [ ] LLM summary includes actionable insights
- [ ] Report delivered to Telegram (text + image attachments)
- [ ] Report stored in Firestore for history
- [ ] Generation completes in <60 seconds

Implementation Checklist:

Backend:

- [ ] Create `ReportAgent` class
- [ ] Implement data aggregation functions (weekly metrics)
- [ ] Create graph generation functions (Plotly)
- [ ] Add Cloud Storage upload for graphs
- [ ] Implement LLM summary generation
- [ ] Create report formatter (Telegram Markdown)
- [ ] Add report history storage in Firestore

Graphs:

- [ ] Implement sleep trends graph
- [ ] Implement workout frequency graph
- [ ] Implement compliance score graph
- [ ] Implement domain radar chart
- [ ] Add graph styling (constitution theme colors)
- [ ] Optimize image size for Telegram (<5MB)

Scheduling:

- [ ] Create Cloud Scheduler job for Sunday 9 AM
- [ ] Implement report generation trigger endpoint
- [ ] Add manual trigger command `/weekly_report`

Bot Integration:

- [ ] Create report message formatter
- [ ] Implement image attachment handler (4 graphs)
- [ ] Add inline buttons (View Dashboard, Export CSV)
- [ ] Handle long messages (split if >4096 chars)

Testing:

- [ ] Unit tests for metric calculations
- [ ] Unit tests for graph generation
- [ ] Integration test for full report flow
- [ ] Test with various data scenarios (low compliance, high compliance)
- [ ] Test LLM summary quality (manual review)

4.4 FEATURE: Monthly Strategic Review

Priority: P1 (High)
Status: Not Started

Owner: TBD
Effort: 3 days

Description:

Comprehensive monthly review generated on the first Sunday of each month (9 AM IST) that assesses goal progress, constitution effectiveness, life trajectory, and deep pattern analysis.

Report Structure:

Section 1: Goal Progress Dashboard

Career Goal: Secure ₹28-42 LPA role by June 2026
- Status: In Progress
- Applications sent: 12 (target: 20/month)
- Interviews: 2
- Offers: 0
- Assessment: ⚠️ Below target, need to increase applications

Physical Goal: Maintain 12% bodyfat, visible abs
- Status: Post-surgery recovery (expected)
- Current: ~15% bodyfat (estimated)
- Workouts: 18/24 completed (75%, maintenance mode)
- Assessment: ✅ On track for surgery recovery phase

Wealth Goal: Rebuild emergency fund to ₹5L post-surgery
- Status: In Progress
- Current savings: ₹2.3L
- Monthly saving rate: ₹40K
- Projected: ₹5L by July 2026
- Assessment: ✅ On track

Relationship Goal: Begin dating May-June 2026
- Status: Active celibacy (by design)
- Duration: 3 months
- Assessment: ✅ Following plan

Section 2: Constitution Effectiveness

What's Working:
✅ Daily check-ins: 87% completion (26/30 days)
✅ Sleep protocol: Avg 7.3hrs (above target)
✅ Pattern detection: Caught 4 violations within 24hrs
✅ Zero porn: 45 consecutive days (new record!)

What's Not Working:
⚠️ Snooze button: Used 12 times this month (target: 0)
⚠️ LeetCode consistency: Only 38 problems (target: 60)
⚠️ Job application rate: 40% below target

What's Missing:
- Need automated LeetCode reminder system
- Consider adding "boredom menu" for consumption triggers
- Add pre-commitment for Sunday job application block

Section 3: Life Trajectory Assessment

30-Year-Old Goals (Target: ~2028):
Goal: ₹2L/month income
- Current: ₹1.67L/month
- Required increase: ₹33K/month (20% raise)
- Timeline: Need new job by June 2026 to stay on track
- Assessment: ✅ Feasible with job switch

Goal: Parents supported significantly
- Current: ₹5K/month support
- Target: ₹15-20K/month
- Gap: ₹10-15K/month
- Assessment: ⚠️ Requires income increase first

Goal: Married with stable relationship
- Current: Single, 18 months to deadline
- Timeline: Tight (6-9 months dating, 3-6 months engagement)
- Assessment: ⚠️ Aggressive timeline, may need family involvement

Goal: Peak physique
- Current: Post-surgery recovery
- Assessment: ✅ Will achieve post-recovery (Apr-June 2026)

Section 4: Deep Pattern Analysis (90-day view)

Behavioral Patterns Identified:

Pattern 1: Relationship Stress → Sleep Disruption Cascade
- Detected: 3 occurrences in last 90 days
- Trigger: Emotional stress (breakup, surgery anxiety)
- Cascade: Reduced sleep → missed workouts → productivity drop

- Constitution response: AI flagged within 24hrs, intervention effective
- Recommendation: Continue emotional processing sessions

Pattern 2: Surgery Anxiety → Procrastination
- Detected: Pre-surgery period (Jan-Feb)
- Trigger: Fear of surgery outcomes
- Behavior: Avoided surgeon confirmation, delayed prep tasks
- Constitution response: AI pushed through resistance, tasks completed
- Recommendation: Trust the intervention system for future major decisions

Pattern 3: Post-Breakup "Just One More Time" Vulnerability
- Detected: First 2 weeks post-breakup
- Trigger: Loneliness + boredom
- High-risk times: Late night (10 PM - 12 AM)
- Constitution response: Interrupt patterns worked (porn-free 45 days)
- Recommendation: Maintain late-night phone blocking, continue streak

Section 5: Course Corrections Needed

Immediate (This Month):
1. Increase job applications to 5/week minimum
   - Action: Block calendar every Wed 7-9 PM, Sat 10-12 AM
   - Accountability: AI reminds Tue/Fri, tracks count

2. Fix snooze button pattern
   - Action: Move phone to bathroom (not just across room)
   - Add math problem alarm (can't dismiss easily)

3. Boost LeetCode consistency
   - Action: Morning slot (6-7 AM) instead of evening
   - Track: AI confirms completion in daily check-in

Strategic (Next Quarter):
1. Prepare for job interview intensity (Mar-May 2026)
   - Build mock interview routine with peers
   - Polish portfolio projects

2. Post-surgery return to peak physique (Apr-June 2026)
   - Gradual progression per constitution protocols
   - Document recovery (before/after photos)

3. Dating preparation (Apr-May 2026)
   - Process relationship trauma fully
   - Wardrobe upgrade (budget ₹20K)

  - Social circle expansion

Acceptance Criteria:

- [ ] Report generated first Sunday of month, 9 AM IST
- [ ] All 5 sections populated with accurate data
- [ ] 90-day pattern analysis uses real data
- [ ] LLM-generated insights are actionable
- [ ] Report delivered to Telegram (may be multiple messages)
- [ ] Includes interactive element (user reflection prompts)
- [ ] Report stored in Firestore for history

Implementation Checklist:

Backend:

- [ ] Create `MonthlyReviewAgent` class
- [ ] Implement goal progress tracker (parse user-defined goals)
- [ ] Add 90-day data aggregation
- [ ] Create deep pattern analysis algorithm
- [ ] Implement trajectory assessment logic
- [ ] Add LLM strategic summary generation
- [ ] Create course correction suggestion system

Data Analysis:

- [ ] Implement goal definition schema
- [ ] Add progress calculation for each goal type
- [ ] Create trajectory projection functions
- [ ] Implement pattern correlation analysis
- [ ] Add effectiveness scoring for constitution protocols

Bot Integration:

- [ ] Create multi-message formatter (report may be long)
- [ ] Add interactive reflection prompts
- [ ] Implement inline keyboard for "Course Correction Accepted"
- [ ] Add export functionality (full report as PDF)

Scheduling:

- [ ] Create Cloud Scheduler job (first Sunday logic)
- [ ] Implement cron expression: `0 9 1-7 * 0` (first Sunday)
- [ ] Add manual trigger command `/monthly_review`

Testing:

- [ ] Unit tests for goal progress calculations
- [ ] Unit tests for pattern correlation analysis
- [ ] Integration test for full review flow
- [ ] Test with 90 days of mock data
- [ ] Test LLM summary quality (manual review)

4.5 FEATURE: Emotional Processing Agent

Priority: P2 (Medium)
Status: Not Started
Owner: TBD
Effort: 2 days

Description:

On-demand CBT-style emotional processing sessions triggered by user when experiencing difficult emotions (loneliness, sadness, anger, urges).

User Flow:
1. User: "I'm feeling lonely and want to look at porn"
2. Bot: "I hear you. Let's process this together. First, what exactly are you feeling? (lonely, sad, anxious, angry, other?)"
3. User: "Lonely and a bit horny"
4. Bot: "Thanks for being honest. What situation triggered this feeling?"
5. User: "Just got home from work, flatmate is out, apartment is quiet, reminded me of ex"
6. Bot: "What thought is your mind offering you right now?"
7. User: "Maybe just one time won't hurt. I deserve some comfort."
8. Bot: [CBT reframe] "Let's reality-check that thought:
   - 'Just one time': Constitution shows you've said this 47 times over last 2 years. It's never 'just once' - it always leads to a pattern.
   - 'Deserve comfort': You DO deserve comfort, but porn isn't comfort - it's temporary numbing followed by shame. Real comfort comes from connection (call a friend, go to café) or processing (which we're doing now).
   - Your emotion is VALID (loneliness is real post-breakup). But porn won't solve loneliness - it'll make it worse.

   Constitution protocol: When lonely → text friend, go to public space.

   What's ONE action you can take in the next 5 minutes that aligns with the person you're becoming?"
9. User: "I could call [friend] or go to the gym"

10. Bot: "Both excellent choices. Which feels more doable right now?"
11. User: "Gym"
12. Bot: "Perfect. I'm setting a 10-minute timer. Go to your room, change into gym clothes, and leave the apartment. Text me when you're there. You've got this. 💪"
[10 minutes later]
13. User: "At gym"
14. Bot: "Proud of you. That was the hard part. Now enjoy the workout - you just chose creation over consumption. That's a WIN. This session logged. Your 45-day streak continues. 🎉"

CBT Techniques Implemented:

1. Cognitive Restructuring

- Identify automatic thought
- Challenge with evidence
- Generate alternative thought

2. Behavioral Activation

- Suggest immediate action
- Provide accountability
- Celebrate follow-through

3. Urge Surfing

- Acknowledge urge without judgment
- Remind of temporary nature
- Redirect to healthier coping

4. Reality Testing

- Reference historical patterns ("last time you did X, result was Y")
- Constitution principles
- Long-term vs short-term thinking

Session Types:

Type 1: Porn Urge Interrupt

- Trigger: User mentions porn/urge
- Focus: Immediate redirect
- Duration: 5-10 min
- Outcome: Action commitment

Type 2: Loneliness Processing

- Trigger: User mentions loneliness/sadness
- Focus: Validate emotion + suggest connection
- Duration: 10-15 min
- Outcome: Social action or self-compassion

Type 3: Relationship Rumination

- Trigger: User mentions ex/relationship
- Focus: Acceptance + forward focus
- Duration: 15-20 min
- Outcome: Closure or reduced rumination

Type 4: Decision Anxiety

- Trigger: User torn about major decision
- Focus: Evidence gathering + pros/cons
- Duration: 20-30 min
- Outcome: Clarity or 48hr research commitment

Acceptance Criteria:

- [ ] Sessions triggered by user message (keyword detection)
- [ ] LLM provides CBT-style reframes (not generic advice)
- [ ] Sessions reference constitution principles
- [ ] Sessions logged in Firestore with outcomes
- [ ] Follow-up check in 24 hours ("How did it go?")
- [ ] Effectiveness tracked (did user follow action?)

Implementation Checklist:

Backend:

- [ ] Create `EmotionalProcessingAgent` class
- [ ] Implement session state management (multi-turn conversation)
- [ ] Add emotion classification (keyword detection)
- [ ] Create CBT prompt templates for each session type
- [ ] Implement constitution reference system (pull relevant principles)
- [ ] Add session logging to Firestore
- [ ] Create follow-up scheduling system (24hr check-in)

LLM Integration:

- [ ] Design CBT prompt templates
- [ ] Add few-shot examples for better responses
- [ ] Implement tone guardrails (supportive but firm)
- [ ] Add constitutional grounding (always reference constitution)

Bot Integration:

- [ ] Create keyword triggers ("lonely", "sad", "porn", "urge", "ex")
- [ ] Implement session initiation flow
- [ ] Add session state persistence (survive bot restart)
- [ ] Create quick action buttons (e.g., "Call friend", "Go to gym")

Analytics:

- [ ] Track session completion rate
- [ ] Track action follow-through rate
- [ ] Analyze most common triggers
- [ ] Measure session effectiveness (did behavior improve?)

Testing:

- [ ] Unit tests for emotion classification
- [ ] Integration tests for each session type
- [ ] Test LLM response quality (manual review)
- [ ] Test multi-turn conversation persistence
- [ ] Test follow-up scheduling

4.6 FEATURE: Crisis/Ghosting Detection

Priority: P2 (Medium)
Status: Not Started
Owner: TBD
Effort: 1 day

Description:

Automated monitoring system that detects when user has missed 3+ consecutive daily check-ins (ghosting), escalates messaging, and prepares for emergency contact notification (future feature).

Ghosting Detection Logic:

```
def check_for_ghosting(user_id: str) -> GhostingStatus:
    """
    Check if user has missed check-ins

    Returns:
    - days_missed: int (consecutive days)
    - last_checkin: datetime
    - status: 'active' | 'warning' | 'critical' | 'emergency'
    """
    last_checkin = get_last_checkin_date(user_id)
    days_missed = (datetime.now(IST) - last_checkin).days

    if days_missed <= 1:
        return {'status': 'active', 'days_missed': 0}
    elif days_missed == 2:
        return {'status': 'warning', 'days_missed': 2}
    elif days_missed <= 4:
        return {'status': 'critical', 'days_missed': days_missed}
    else:
        return {'status': 'emergency', 'days_missed': days_missed}
```

Escalation Protocol:

Day 1: Missed Check-in

- **Action:** None (user might be busy)
- **Message:** None

Day 2: 1 Consecutive Miss

- **Action:** Gentle reminder
- **Message:** "Hey, I noticed you missed yesterday's check-in. Everything okay? Remember, consistency is key. Ready to check in now?"

Day 3: 2 Consecutive Misses

- **Action:** Urgent reminder
- **Message:** "⚠️ You've missed 2 check-ins in a row. This is unusual for you. Are you in crisis? Your longest streak was 47 days - don't break it now. Respond ASAP."

Day 4: 3 Consecutive Misses

- **Action:** Pattern analysis + motivational message
- **Message:** "🚨 3 days without check-in. This is a red flag. When you ghosted systems in the past, it was because: [relationship consuming you / depressive episode / avoiding accountability]. Which is it? I'm here to help, not judge. Your constitution needs you. Respond."

Day 5: 4 Consecutive Misses

- **Action:** Emergency protocol consideration

- **Message:** "🆘 Day 5 of silence. This is concerning. Your constitution says I should reach out to an accountability partner at this point. I haven't done that yet, but if you don't respond by tomorrow, I may need to take action. Please check in."
- **Future:** Message accountability partner (feature parked for now)

Day 7+: Extended Ghosting

- **Action:** Log emergency status
- **Message:** Daily reminders continue
- **Future:** Send emergency resources (therapy hotlines, etc.)

Return Protocol:

When user returns after ghosting:

1. Bot: "Welcome back! I noticed you were gone for {days} days. I'm glad you're here. Before we resume, can you share what happened? (This helps me support you better)"
2. User: [Explains reason]
3. Bot: [Acknowledges without judgment]
   - If crisis: "That sounds really tough. Thank you for coming back. Constitution says: restart without guilt. Your streak resets, but your progress doesn't. Let's rebuild."
   - If avoidance: "I hear you. Avoiding accountability is a pattern we've identified. The fact that you came back shows strength. Let's put a system in place to prevent this next time."
4. Bot: "Ready to do today's check-in? We'll start fresh from here."
5. [Resume normal check-in flow]

Acceptance Criteria:

- [ ] Ghosting check runs daily at 10 PM IST
- [ ] Escalation messages sent at correct intervals
- [ ] Ghosting status logged in Firestore
- [ ] Return protocol executes when user comes back
- [ ] Streak resets appropriately after ghosting
- [ ] Emergency contact feature architected (but not implemented yet)

Implementation Checklist:

Backend:

- [ ] Create `GhostingDetectionAgent` class
- [ ] Implement ghosting check logic
- [ ] Add escalation message templates (Day 2-7+)
- [ ] Create return protocol flow
- [ ] Add Firestore logging for ghosting events
- [ ] Implement streak reset logic

Bot Integration:

- [ ] Create automated reminder sender
- [ ] Implement escalation timing logic
- [ ] Add return welcome message
- [ ] Create "comeback" flow (gentle reentry)

Scheduling:

- [ ] Create Cloud Scheduler job for 10 PM IST
- [ ] Implement ghosting check trigger endpoint

Analytics:

- [ ] Track ghosting frequency
- [ ] Analyze ghosting triggers (patterns before ghosting)
- [ ] Measure return rate (% who come back)

Testing:

- [ ] Unit tests for ghosting detection logic
- [ ] Test escalation timing
- [ ] Test return protocol
- [ ] Simulate 7-day ghosting scenario

4.7 FEATURE: Streamlit Dashboard

Priority: P3 (Low - Nice to Have)
Status: Not Started
Owner: TBD
Effort: 3 days

Description:

Web-based dashboard (Streamlit) for visualizing constitution compliance, viewing historical data, generating custom reports, and exporting data.

Dashboard Pages:

Page 1: Overview Dashboard

- Date range selector (7/30/90/365 days)
- Key metrics cards (streak, compliance %, sleep avg, workouts)
- Compliance trend graph

- Domain radar chart
- Recent patterns list

Page 2: Physical Health

- Sleep trend graph (with 7hr target line)
- Workout frequency calendar heatmap
- Body metrics tracking (weight, bodyfat % - manual input)
- Training split visualization

Page 3: Career Progress

- LeetCode progress graph (daily/weekly)
- Job application funnel (applied → interview → offer)
- Deep work hours trend
- Skill development tracking

Page 4: Mental Health

- Check-in completion calendar
- Mood rating trend
- Emotional processing session log
- Pattern detection history

Page 5: Wealth

- Net worth tracking graph
- Monthly budget vs actual
- SIP contribution tracking
- Savings rate trend

Page 6: Data Export

- Export check-in data (CSV)
- Export metrics (CSV)
- Generate custom report (PDF)
- Download graphs (PNG)

Technical Architecture:

```
# app.py (Streamlit entry point)
import streamlit as st
from google.cloud import firestore

st.set_page_config(layout="wide", page_title="Constitution Dashboard")

# Sidebar
```

```
st.sidebar.title("Constitution Dashboard")
user_id = st.sidebar.text_input("User ID", value="ayush_default")
date_range = st.sidebar.selectbox("Date Range", ["7 days", "30 days", "90 days", "365 days"])

# Main content
st.title("Constitution Compliance Dashboard")

# Key Metrics
col1, col2, col3, col4 = st.columns(4)
col1.metric("Streak Days", data['streak'], f"+{data['streak_change']}")
col2.metric("Compliance", f"{data['compliance']}%", f"{data['compliance_change']}%")
col3.metric("Sleep Avg", f"{data['sleep_avg']}hrs", f"{data['sleep_change']}hrs")
col4.metric("Workouts", f"{data['workouts']}/{data['workout_target']}", "On track")

# Graphs
st.plotly_chart(generate_compliance_trend_graph(data))
st.plotly_chart(generate_domain_radar_chart(data))

# Patterns
st.subheader("Active Patterns")
for pattern in data['patterns']:
    st.warning(f"⚠️ {pattern['description']}")
```

Deployment:

Option 1: Same Cloud Run instance as bot

- Streamlit runs on separate port
- Shared Firestore connection
- Cost: $0 (within free tier)

Option 2: Separate Cloud Run service

- Dedicated instance for dashboard
- Independent scaling
- Cost: ~$0.20/month (minimal usage)

Recommendation: Option 1 (same instance) for cost optimization

Acceptance Criteria:

- [ ] Dashboard accessible via URL (authenticated)
- [ ] All 6 pages functional
- [ ] Graphs interactive (Plotly)
- [ ] Real-time data from Firestore

- [ ] Export functions work (CSV/PDF)
- [ ] Mobile-responsive design
- [ ] Load time <3 seconds

Implementation Checklist:

Backend:

- [ ] Create Streamlit app structure (6 pages)
- [ ] Implement Firestore data fetching
- [ ] Create graph generation functions (reuse from reports)
- [ ] Add data export functions (CSV/PDF)
- [ ] Implement authentication (simple password or Google SSO)

Pages:

- [ ] Build Overview page
- [ ] Build Physical Health page
- [ ] Build Career Progress page
- [ ] Build Mental Health page
- [ ] Build Wealth page
- [ ] Build Data Export page

Deployment:

- [ ] Containerize Streamlit app (Dockerfile)
- [ ] Deploy to Cloud Run
- [ ] Configure custom domain (optional)
- [ ] Set up authentication

Testing:

- [ ] Test all pages with real data
- [ ] Test export functions
- [ ] Test on mobile devices
- [ ] Load test (simulate multiple users)

---

5. DATA MODELS

5.1 Firestore Schema

Collection: `users`

Document ID: `{user_id}` (e.g., "ayush_primary")

```
{
  "user_id": "ayush_primary",
  "telegram_id": 123456789,
  "telegram_username": "ayush_username",
  "name": "Ayush",
  "email": "ayush@example.com",
  "timezone": "Asia/Kolkata",

  "constitution": {
    "version": "1.0",
    "current_mode": "maintenance",
    "mode_history": [
      {"mode": "optimization", "start_date": "2026-01-01", "end_date": "2026-01-26"},
      {"mode": "maintenance", "start_date": "2026-01-27", "end_date": null}
    ]
  },

  "streaks": {
    "current_streak": 47,
    "longest_streak": 47,
    "last_checkin_date": "2026-01-29",
    "total_checkins": 147
  },

  "goals": {
    "career": {
      "target_salary": 3200000,
      "target_date": "2026-06-30",
      "status": "in_progress"
    },
    "physical": {
      "target_bodyfat": 12,
      "target_weight": 180,
      "status": "maintenance"
    }
  },

  "preferences": {
    "check_in_time": "21:00",
    "reminder_enabled": true,
    "dashboard_access": true
  },
  "metadata": {
    "created_at": "2026-01-27T00:00:00Z",
    "updated_at": "2026-01-29T15:30:00Z",
    "last_active": "2026-01-29T21:15:00Z"
  }
}
```

Collection: `daily_checkins/{user_id}/checkins`

Document ID: `{date}` (e.g., "2026-01-29")

```
{
  "date": "2026-01-29",
  "user_id": "ayush_primary",
  "mode": "maintenance",

  "tier1_non_negotiables": {
    "sleep": {
      "completed": true,
      "hours": 7.5,
      "quality": "good"
    },
    "training": {
      "completed": true,
      "type": "workout",
      "is_rest_day": false
    },
    "deep_work": {
      "completed": true,
      "hours": 2.5
    },
    "zero_porn": {
      "completed": true
    },
    "boundaries": {
      "completed": true,
      "notes": null
    }
  },

  "responses": {
    "challenges": "Felt a bit lonely after work, but went to gym instead of staying home",
    "rating": 8,
    "rating_reason": "Solid day overall, handled loneliness well",
    "tomorrow_priority": "Complete system design chapter on distributed caching",
    "tomorrow_obstacle": "Might have late meeting, need to block evening time"
  },

  "computed": {
    "compliance_score": 100.0,
    "domain_scores": {
      "physical": 100.0,
      "career": 100.0,
      "mental": 100.0,
      "boundaries": 100.0
    }
  },

  "metadata": {
    "completed_at": "2026-01-29T21:15:42Z",
    "duration_seconds": 180,
    "reminder_sent": true,
    "reminder_count": 1
  }
}
```

Collection: `metrics/{user_id}/weekly`

Document ID: `{week_id}` (e.g., "2026-W05")

```
{
  "week_id": "2026-W05",
  "user_id": "ayush_primary",
  "week_start": "2026-01-27",
  "week_end": "2026-02-02",
  "mode": "maintenance",

  "physical": {
    "workouts_completed": 5,
    "workouts_target": 6,
    "workout_completion_rate": 0.83,
    "sleep_avg": 7.2,
    "sleep_target": 7.0,
    "protein_avg": 165,
    "protein_target": 180
  },

  "career": {
    "leetcode_problems": 12,
    "leetcode_target": 14,
    "job_applications": 3,
    "job_applications_target": 5,
    "deep_work_hours_avg": 2.3,
    "deep_work_target": 2.0
  },

  "mental": {
    "checkins_completed": 7,
    "checkins_target": 7,
    "checkin_completion_rate": 1.0,
    "emotional_sessions": 2,
    "mood_ratings": [8, 7, 9, 6, 8, 7, 9],
    "mood_avg": 7.71
  },

  "wealth": {
    "sip_deposited": true,
    "sip_amount": 30000,
    "budget_adherence": 0.95,
    "total_spent": 158000
  },

  "compliance": {
    "overall_score": 85.0,
    "domain_scores": {
      "physical": 83.0,
      "career": 78.0,
      "mental": 100.0,
      "wealth": 95.0
    }
  },

  "metadata": {
    "generated_at": "2026-02-03T09:00:00Z",
    "report_sent": true
  }
}
```

Collection: `patterns/{user_id}/detected`

Document ID: Auto-generated

```
{
  "pattern_id": "pattern_abc123",
  "user_id": "ayush_primary",
  "pattern_type": "sleep_degradation",
  "severity": "warning",

  "detection": {
    "detected_at": "2026-01-29T22:00:00Z",
    "detection_method": "rule_based",
    "rule_triggered": "sleep_hours < 6 for 3+ consecutive days"
  },

  "details": {
    "description": "Sleep <6 hours for 3 consecutive nights",
    "trigger_events": [
      {"date": "2026-01-27", "sleep_hours": 5.5, "reason": "late call"},
      {"date": "2026-01-28", "sleep_hours": 5.8, "reason": "work stress"},
      {"date": "2026-01-29", "sleep_hours": 5.2, "reason": "doom scrolling"}
    ],
    "historical_context": "Last time this pattern occurred: Feb 2025, led to 6-month training regression"
  },

  "intervention": {
    "intervention_triggered": true,
    "intervention_id": "intervention_xyz789",
    "intervention_level": "warning",
    "message_sent": true,
    "user_response": "Acknowledged, will fix tonight",
    "effective": true
  },

  "resolution": {
    "resolved": true,
    "resolved_at": "2026-01-30T06:30:00Z",
    "resolution_method": "behavior_corrected",
    "sleep_hours_post": 7.5,
    "notes": "User corrected sleep same night after warning"
  },
  "metadata": {
    "created_at": "2026-01-29T22:00:00Z",
    "updated_at": "2026-01-30T06:30:00Z"
  }
}
```

Collection: `interventions/{user_id}/history`

Document ID: Auto-generated

```
{
  "intervention_id": "intervention_xyz789",
  "user_id": "ayush_primary",
  "pattern_id": "pattern_abc123",

  "intervention_type": "warning",
  "severity": "warning",

  "trigger": {
    "triggered_by": "pattern_detection",
    "pattern_type": "sleep_degradation",
    "triggered_at": "2026-01-29T22:00:00Z"
  },

  "message": {
    "template": "sleep_degradation_warning",
    "content": "You've violated sleep protocol 3 nights in a row. Constitution says: sleep is non-negotiable. You're entering spiral pattern that led to 6-month regression in Feb 2025. We need to fix this TODAY. What's blocking you?",
    "tone": "firm",
    "references_constitution": true
  },

  "delivery": {
    "sent_at": "2026-01-29T22:01:15Z",
    "channel": "telegram",
    "delivery_status": "delivered",
    "read_at": "2026-01-29T22:03:42Z"
  },

  "user_response": {
    "responded": true,
    "response_time_seconds": 180,
    "response_text": "Acknowledged, will fix tonight",
    "sentiment": "accepting"
  },

  "effectiveness": {
    "effective": true,
    "behavior_corrected": true,
    "correction_time_hours": 8,
    "post_intervention_metric": {
      "sleep_hours": 7.5,
      "meets_target": true
    }
  },

  "metadata": {
    "created_at": "2026-01-29T22:00:00Z",
    "updated_at": "2026-01-30T06:30:00Z"
  }
}
```

Collection: `emotional_sessions/{user_id}/sessions`

Document ID: Auto-generated

```
{
  "session_id": "session_def456",
  "user_id": "ayush_primary",

  "trigger": {
    "triggered_by": "user_message",
    "trigger_keywords": ["lonely", "porn"],
    "trigger_message": "I'm feeling lonely and want to look at porn",
    "triggered_at": "2026-01-29T19:30:00Z"
  },

  "session_type": "porn_urge_interrupt",

  "conversation": [
    {
      "role": "bot",
      "message": "I hear you. Let's process this together. First, what exactly are you feeling?",
      "timestamp": "2026-01-29T19:30:05Z"
    },
    {
      "role": "user",
      "message": "Lonely and a bit horny",
      "timestamp": "2026-01-29T19:30:42Z"
    },
    {
      "role": "bot",
      "message": "Thanks for being honest. What situation triggered this feeling?",
      "timestamp": "2026-01-29T19:30:47Z"
    }
  ],

  "analysis": {
    "primary_emotion": "loneliness",
    "secondary_emotion": "sexual_desire",
    "cbt_technique_used": "cognitive_restructuring",
    "constitution_reference": "When lonely → text friend, go to public space",
    "alternative_thought": "Porn won't solve loneliness - it'll make it worse. Real comfort comes from connection."
  },

  "outcome": {
    "action_committed": "go_to_gym",
    "action_followed_through": true,
    "followup_scheduled": "2026-01-30T19:30:00Z",
    "followup_completed": true,
    "session_effective": true
  },

  "metadata": {
    "started_at": "2026-01-29T19:30:00Z",
    "ended_at": "2026-01-29T19:45:32Z",
    "duration_seconds": 932,
    "message_count": 12
  }
}
```

5.2 Data Flow Diagrams

Flow 1: Daily Check-In

User → Bot: "Ready for check-in"
↓
Bot → LangGraph: Start CheckInAgent
↓

CheckInAgent → User: Ask Question 1 (Tier 1 non-negotiables)
↓
User → CheckInAgent: Respond with Y/N for each item
↓
CheckInAgent: Parse & validate responses
↓
CheckInAgent → Firestore: Store partial check-in
↓
CheckInAgent → User: Ask Question 2 (Challenges)
↓
User → CheckInAgent: Free text response
↓
[Continue for Questions 3 & 4]
↓
CheckInAgent: Calculate compliance score
↓
CheckInAgent → Firestore: Store complete check-in
↓
CheckInAgent → PatternDetectionAgent: Trigger pattern scan
↓
PatternDetectionAgent: Scan for violations
↓
PatternDetectionAgent → InterventionAgent (if pattern detected)
↓
CheckInAgent → LLM: Generate feedback summary
↓
CheckInAgent → User: Send feedback (streak, score, insights)
↓
User: Check-in complete

Flow 2: Pattern Detection & Intervention

Cloud Scheduler (6hr intervals) → FastAPI: Trigger pattern scan
↓
FastAPI → PatternDetectionAgent: Start scan for user
↓
PatternDetectionAgent → Firestore: Fetch last 7 days check-ins
↓
PatternDetectionAgent: Run rule engine on data
↓
PatternDetectionAgent: Detected pattern? (Y/N)
↓ (if Yes)
PatternDetectionAgent → Firestore: Store pattern
↓

PatternDetectionAgent: Assess severity (nudge/warning/veto)
↓
PatternDetectionAgent → InterventionAgent: Trigger intervention
↓
InterventionAgent: Select message template based on severity
↓
InterventionAgent → LLM: Generate personalized message
↓
InterventionAgent → Firestore: Store intervention
↓
InterventionAgent → Bot: Send message to user
↓
Bot → User: Intervention message
↓
User → Bot: Response (acknowledge/dispute/ignore)
↓
Bot → InterventionAgent: Log user response
↓
InterventionAgent → Firestore: Update intervention with response
↓
[24-48 hours later]
↓
PatternDetectionAgent: Check if behavior corrected
↓
PatternDetectionAgent → Firestore: Update pattern resolution + intervention effectiveness

Flow 3: Weekly Report Generation

Cloud Scheduler (Sunday 9 AM) → FastAPI: Trigger weekly report
↓
FastAPI → ReportAgent: Start report generation
↓
ReportAgent → Firestore: Fetch last 7 days data
↓
ReportAgent: Aggregate metrics (calculate averages, counts)
↓
ReportAgent: Calculate compliance scores by domain
↓
ReportAgent: Generate 4 graphs (Plotly)
├── Sleep trend graph
├── Workout frequency graph
├── Compliance score graph
└── Domain radar chart
↓

ReportAgent → Cloud Storage: Upload graphs (PNG)
↓
ReportAgent → LLM: Generate narrative summary
↓
LLM: Returns 200-300 word summary
↓
ReportAgent: Format report (Markdown + images)
↓
ReportAgent → Firestore: Store report + metrics
↓
ReportAgent → Bot: Send report to user (text + 4 images)
↓
Bot → User: Weekly report delivered
↓
User: Reads report, provides feedback (optional)

---

6. API SPECIFICATIONS

6.1 Telegram Bot Commands

Command	Description	Parameters	Example
`/start`	Initialize bot, register user	None	`/start`
`/checkin`	Trigger daily check-in manually	None	`/checkin`
`/status`	View current streak & compliance	None	`/status`
`/weekly`	Generate weekly report manually	None	`/weekly`
`/monthly`	Generate monthly review manually	None	`/monthly`
`/help`	Show available commands	None	`/help`
`/mode`	View/change constitution mode	`[optimization/maintenance/survival]`	`/mode maintenance`

`/dashboard`	Get dashboard URL	None	`/dashboard`
`/export`	Export data as CSV	`[checkins/metrics/all]`	`/export checkins`
`/settings`	View/change preferences	None	`/settings`

6.2 Telegram Bot Message Types

Incoming Messages:

```
# Text message from user
{
  "message_id": 12345,
  "from": {"id": 123456789, "username": "ayush_username"},
  "chat": {"id": 123456789, "type": "private"},
  "text": "I'm feeling lonely",
  "date": 1706543210
}

# Callback query (button press)
{
  "callback_query": {
    "id": "query_abc123",
    "from": {"id": 123456789},
    "message": {...},
    "data": "sleep_yes"
  }
}
```

Outgoing Messages:

```
# Text message to user
{
  "chat_id": 123456789,
  "text": "Time for your daily check-in! Ready?",
  "parse_mode": "Markdown"
}

# Message with inline keyboard
{
  "chat_id": 123456789,
  "text": "Sleep: 7+ hours last night?",
  "reply_markup": {
    "inline_keyboard": [
      [
        {"text": "✅ Yes", "callback_data": "sleep_yes"},
        {"text": "❌ No", "callback_data": "sleep_no"}
      ]
    ]
  }
}

# Photo message (for graphs)
{
  "chat_id": 123456789,
  "photo": "https://storage.googleapis.com/.../sleep_graph.png",
  "caption": "Your sleep trends over the past 4 weeks"
}
```

6.3 FastAPI Endpoints

Webhook Endpoint (Telegram)

```
POST /webhook/telegram
Headers:
  - Content-Type: application/json
  - X-Telegram-Bot-Api-Secret-Token: <secret>

Request Body:
{
  "update_id": 123456,
  "message": {
    "message_id": 789,
    "from": {"id": 123456789, "username": "ayush_username"},
    "chat": {"id": 123456789, "type": "private"},
    "text": "/checkin",
    "date": 1706543210
  }
}

Response:
Status: 200 OK
Body: {"ok": true}
```

Scheduled Trigger Endpoints

```
# Daily check-in trigger
POST /trigger/daily-checkin
Headers:
  - X-CloudScheduler-JobName: daily-checkin-trigger

Response:
Status: 200 OK
Body: {"status": "triggered", "users": ["ayush_primary"]}

# Weekly report trigger
POST /trigger/weekly-report
Headers:
  - X-CloudScheduler-JobName: weekly-report-trigger

Response:
Status: 200 OK
Body: {"status": "generated", "users": ["ayush_primary"]}

# Pattern scan trigger
POST /trigger/pattern-scan
Headers:
  - X-CloudScheduler-JobName: pattern-scan-trigger

Response:
Status: 200 OK
Body: {"status": "scanned", "patterns_detected": 2}
```

Dashboard Authentication

```
# Simple password auth (for MVP)
POST /auth/login
Request Body:
{
  "password": "constitution_secure_password"
}

Response:
Status: 200 OK
Body: {"token": "jwt_token_here"}

# Access dashboard
GET /dashboard
Headers:
  - Authorization: Bearer <jwt_token>

Response:
Status: 200 OK
Body: HTML (Streamlit app)
```

6.4 Vertex AI (Gemini) API Calls

```
# Intent classification
from vertexai.generative_models import GenerativeModel

model = GenerativeModel("gemini-2.0-flash-exp")

prompt = """
Classify the user's intent from this message:
User message: "I'm feeling lonely and want to look at porn"

Possible intents:
- checkin (daily check-in request)
- emotional (emotional processing needed)
- query (general question)
- command (bot command like /status)

Response format: {"intent": "<intent>", "confidence": <0-1>}
"""

response = model.generate_content(prompt)
# Output: {"intent": "emotional", "confidence": 0.95}

# Check-in feedback generation
prompt = f"""
You are Ayush's constitution AI. Generate feedback for his daily check-in.

Data:
- Compliance score: {compliance_score}%
- Streak: {streak} days
- Sleep: {sleep_hours} hrs
- Workout: {'Completed' if workout else 'Missed'}
- Patterns detected: {patterns}

Constitution mode: {mode}

Guidelines:
- Supportive but firm tone
- Reference constitution when relevant
- Keep it 3-4 sentences
- End with encouragement or challenge

Generate feedback:
"""

response = model.generate_content(prompt)
# Output: "Solid work today - 9/10 constitution alignment..."
```

---

7. IMPLEMENTATION PHASES

PHASE 1: MVP - Basic Check-In System (Week 1)

Goal: Get daily check-ins working end-to-end

Estimated Effort: 5-7 days
Status: Not Started

Tasks:

1.1 Project Setup

- [ ] Create GCP project
- [ ] Enable required APIs (Cloud Run, Firestore, Vertex AI, Cloud Scheduler, Cloud Storage)
- [ ] Set up Firestore database (Native mode, asia-south1)
- [ ] Create Cloud Storage bucket for graphs
- [ ] Set up local development environment
    - [ ] Install Python 3.11+
    - [ ] Install dependencies (requirements.txt)
    - [ ] Configure GCP credentials (service account key)
- [ ] Initialize Git repository
- [ ] Create GitHub repo (private)
- [ ] Set up .gitignore (exclude credentials, **pycache**, etc.)

1.2 Telegram Bot Setup

- [ ] Create Telegram bot via BotFather
- [ ] Get bot token
- [ ] Configure webhook URL (Cloud Run URL + /webhook/telegram)
- [ ] Test bot responds to `/start`

- [ ] Implement basic command handlers
    - [ ] `/start` - Welcome message
    - [ ] `/help` - Show commands
    - [ ] `/checkin` - Trigger check-in
- [ ] Store bot token in GCP Secret Manager

1.3 Firestore Schema Implementation

- [ ] Create `users` collection
- [ ] Create `daily_checkins/{user_id}/checkins` subcollection
- [ ] Write helper functions for Firestore CRUD
    - [ ] `create_user(user_data)`
    - [ ] `get_user(user_id)`
    - [ ] `update_user(user_id, updates)`
    - [ ] `store_checkin(user_id, checkin_data)`
    - [ ] `get_checkin(user_id, date)`
    - [ ] `get_recent_checkins(user_id, days)`
- [ ] Test Firestore connection
- [ ] Add indexes for common queries

1.4 Basic Check-In Flow (Hardcoded)

- [ ] Create conversational state management
    - [ ] Use python-telegram-bot ConversationHandler
    - [ ] Define states: START, Q1, Q2, Q3, Q4, END
- [ ] Implement Question 1 (Tier 1 non-negotiables)
    - [ ] Send question text
    - [ ] Create inline keyboard (Y/N buttons)
    - [ ] Parse response (handle text "yes/no" or button callback)
    - [ ] Validate all 5 items answered
    - [ ] Store partial response
- [ ] Implement Question 2 (Challenges)
    - [ ] Send question text
    - [ ] Parse free text response
    - [ ] Validate min 10 chars
    - [ ] Store response
- [ ] Implement Question 3 (Rating)
    - [ ] Send question text
    - [ ] Parse rating (1-10) + reason
    - [ ] Validate format
    - [ ] Store response
- [ ] Implement Question 4 (Tomorrow's priority)
    - [ ] Send question text

    - [ ] Parse priority + obstacle
    - [ ] Store response
- [ ] Calculate compliance score
    - [ ] Sum Tier 1 items (5 total)
    - [ ] Calculate percentage
    - [ ] Store in Firestore
- [ ] Send simple feedback
    - [ ] "Check-in complete! Compliance: X%"
    - [ ] "Streak: Y days"
- [ ] Handle timeouts (15 min inactivity → prompt "Still there?")
- [ ] Handle interruptions (user can resume check-in)

1.5 Streak Tracking

- [ ] Implement streak counter logic
    - [ ] Increment on successful check-in
    - [ ] Reset if >48 hours since last check-in
    - [ ] Track longest streak
- [ ] Update user document with streak data
- [ ] Test streak calculation edge cases

1.6 FastAPI Application

- [ ] Create FastAPI app structure
    - [ ] `main.py` (entry point)
    - [ ] `routers/` (webhook, triggers)
    - [ ] `services/` (business logic)
    - [ ] `models/` (Pydantic models)
    - [ ] `utils/` (helpers)
- [ ] Implement `/webhook/telegram` endpoint
    - [ ] Verify Telegram signature
    - [ ] Parse incoming updates
    - [ ] Route to appropriate handler
- [ ] Implement health check endpoint `/health`
- [ ] Add error handling middleware
- [ ] Add request logging

1.7 Deployment (Cloud Run)

- [ ] Create Dockerfile
    - [ ] Base image: python:3.11-slim
    - [ ] Install dependencies
    - [ ] Copy application code
    - [ ] Set entrypoint (uvicorn)

- [ ] Build Docker image
- [ ] Push to Google Container Registry
- [ ] Deploy to Cloud Run
    - [ ] Region: asia-south1
    - [ ] CPU: 1, Memory: 512Mi
    - [ ] Min instances: 0, Max instances: 3
    - [ ] Set environment variables (BOT_TOKEN, GCP_PROJECT, etc.)
- [ ] Configure Telegram webhook to Cloud Run URL
- [ ] Test end-to-end (send /checkin, complete flow)

1.8 Manual Testing

- [ ] Test `/start` command
- [ ] Test `/checkin` command
- [ ] Complete full check-in flow
- [ ] Test Y/N button responses
- [ ] Test free text responses
- [ ] Test timeout handling
- [ ] Test resume after interruption
- [ ] Verify data stored correctly in Firestore
- [ ] Verify streak updates correctly
- [ ] Test error scenarios (invalid input, network failure)

Phase 1 Success Criteria:

- [ ] User can complete daily check-in via Telegram
- [ ] All 4 questions asked and answered
- [ ] Responses stored in Firestore
- [ ] Compliance score calculated correctly
- [ ] Streak tracked accurately
- [ ] System responds in <5 seconds
- [ ] No critical bugs in happy path

PHASE 2: LangGraph + Pattern Detection (Week 2)

Goal: Add multi-agent system and pattern detection

Estimated Effort: 5-7 days
Status: Not Started

Tasks:

2.1 LangGraph Setup

- [ ] Install LangGraph + LangChain dependencies
- [ ] Create LangGraph state schema
    - [ ] Define `ConstitutionState` TypedDict
    - [ ] Fields: user_id, message, intent, context, response, next_action
- [ ] Create base agent class
    - [ ] Common methods: `invoke()`, `ainvoke()`
    - [ ] Firestore access
    - [ ] LLM access
- [ ] Test LangGraph basic flow

2.2 Supervisor Agent

- [ ] Create `SupervisorAgent` class
    - [ ] Intent classification logic
    - [ ] Route to appropriate sub-agent
- [ ] Implement intent classifier
    - [ ] Use Gemini for classification
    - [ ] Intents: checkin, emotional, query, command
    - [ ] Prompt engineering for accurate classification
- [ ] Add routing logic
    - [ ] Map intent → agent
    - [ ] Handle unknown intents
- [ ] Test supervisor with various messages

2.3 Refactor Check-In Agent (LangGraph-based)

- [ ] Create `CheckInAgent` class (extends base agent)
- [ ] Port hardcoded check-in flow to LangGraph
    - [ ] Question flow as graph states
    - [ ] Transitions between questions
- [ ] Add LLM for feedback generation
    - [ ] Replace hardcoded feedback
    - [ ] Use Gemini to generate personalized response
    - [ ] Prompt template with constitution context
- [ ] Test new check-in agent matches old behavior

2.4 Pattern Detection Agent

- [ ] Create `PatternDetectionAgent` class
- [ ] Define pattern rules (Python dict)
    - [ ] Sleep degradation rule
    - [ ] Training abandonment rule
    - [ ] Porn relapse rule
    - [ ] Boundary violation rule

    - [ ] Ghosting rule
- [ ] Implement rule engine
    - [ ] Fetch recent check-ins (7 days)
    - [ ] Evaluate each rule
    - [ ] Return list of detected patterns
- [ ] Add severity assessment
    - [ ] Nudge: 1 violation
    - [ ] Warning: 2-3 violations
    - [ ] Veto: Critical or 4+ violations
- [ ] Store patterns in Firestore
    - [ ] Create `patterns/{user_id}/detected` collection
    - [ ] Document schema
- [ ] Test pattern detection with mock data

2.5 Intervention Agent

- [ ] Create `InterventionAgent` class
- [ ] Implement message template system
    - [ ] Templates for each pattern + severity
    - [ ] Variables: {count}, {days}, {constitution_reference}
- [ ] Add LLM personalization
    - [ ] Take template + context
    - [ ] Generate personalized message
    - [ ] Maintain tone (supportive but firm)
- [ ] Implement delivery logic
    - [ ] Send message via Telegram
    - [ ] Log intervention in Firestore
- [ ] Add effectiveness tracking
    - [ ] Track user response
    - [ ] Check if behavior corrected (24-48hr later)
- [ ] Test interventions with various patterns

2.6 Scheduled Pattern Scanning

- [ ] Create Cloud Scheduler job (every 6 hours)
    - [ ] Cron: `0 */6 * * *`
    - [ ] Target: Cloud Run endpoint `/trigger/pattern-scan`
- [ ] Implement `/trigger/pattern-scan` endpoint
    - [ ] Verify request from Cloud Scheduler
    - [ ] Trigger PatternDetectionAgent for all users
    - [ ] Return scan results
- [ ] Add on-demand scan (after each check-in)
    - [ ] Call PatternDetectionAgent after check-in completes
    - [ ] Immediate pattern detection

- [ ] Test scheduled scanning

2.7 Integration Testing

- [ ] Test Supervisor routing (send various messages)
- [ ] Test CheckInAgent with LLM feedback
- [ ] Test PatternDetectionAgent with 7 days of mock data
- [ ] Test InterventionAgent message generation
- [ ] Test end-to-end: check-in → pattern detected → intervention sent
- [ ] Test scheduled pattern scan
- [ ] Verify all data stored correctly in Firestore

2.8 Vertex AI Optimization

- [ ] Optimize prompts for Gemini Flash (minimize tokens)
- [ ] Add caching for repeated prompts
- [ ] Monitor API costs (ensure <$0.10/day)
- [ ] Test latency (ensure <5 sec responses)

Phase 2 Success Criteria:

- [ ] LangGraph supervisor routes correctly
- [ ] Check-in agent generates LLM-based feedback
- [ ] Pattern detection catches all test cases (9 patterns)
- [ ] Interventions sent for detected patterns
- [ ] Scheduled scans run every 6 hours
- [ ] End-to-end flow works smoothly
- [ ] Cost remains <$0.20/day

PHASE 3: Reports & Dashboard (Week 3)

Goal: Add weekly/monthly reports and web dashboard

Estimated Effort: 5-7 days
Status: Not Started

Tasks:

3.1 Metrics Aggregation

- [ ] Create `metrics/{user_id}/weekly` collection schema
- [ ] Implement weekly aggregation function
    - [ ] Fetch 7 days of check-ins
    - [ ] Calculate domain scores (physical, career, mental, wealth)

    - [ ] Calculate averages (sleep, mood, etc.)
    - [ ] Store in Firestore
- [ ] Implement monthly aggregation function
    - [ ] Fetch 30 days of check-ins
    - [ ] Calculate goal progress
    - [ ] Analyze 90-day patterns
    - [ ] Store in Firestore
- [ ] Test aggregation with mock data

3.2 Report Agent

- [ ] Create `ReportAgent` class
- [ ] Implement weekly report generation
    - [ ] Fetch weekly metrics
    - [ ] Generate 4 graphs (Plotly)
    - [ ] Upload graphs to Cloud Storage
    - [ ] Generate LLM summary (200-300 words)
    - [ ] Format report (Markdown)
- [ ] Implement monthly report generation
    - [ ] Fetch monthly metrics
    - [ ] Generate goal progress section
    - [ ] Generate deep pattern analysis (LLM)
    - [ ] Generate course corrections (LLM)
    - [ ] Format report
- [ ] Test report generation with mock data

3.3 Graph Generation

- [ ] Install Plotly + dependencies
- [ ] Implement sleep trend graph
    - [ ] Line chart, 28 days
    - [ ] Target line at 7 hrs
    - [ ] Color zones (green/yellow/red)
- [ ] Implement workout frequency graph
    - [ ] Bar chart, 4 weeks
    - [ ] Target line at 6
- [ ] Implement compliance score graph
    - [ ] Line chart, 28 days
    - [ ] Moving average (7-day)
- [ ] Implement domain radar chart
    - [ ] Polar chart
    - [ ] Current week vs last week
- [ ] Style graphs (constitution theme colors)
- [ ] Optimize image size (<500KB per graph)
- [ ] Test graph generation with various data

3.4 Cloud Scheduler for Reports

- [ ] Create weekly report job
    - [ ] Cron: `0 9 * * 0` (Sunday 9 AM IST)
    - [ ] Target: `/trigger/weekly-report`
- [ ] Create monthly report job
    - [ ] Cron: `0 9 1-7 * 0` (First Sunday 9 AM IST)
    - [ ] Target: `/trigger/monthly-review`
- [ ] Implement trigger endpoints
    - [ ] Verify Cloud Scheduler signature
    - [ ] Trigger ReportAgent
    - [ ] Send report to user via Telegram
- [ ] Test scheduled report generation

3.5 Report Delivery (Telegram)

- [ ] Implement multi-message sender (if report >4096 chars)
- [ ] Implement image attachment handler
    - [ ] Send 4 graphs as photo messages
    - [ ] Add captions
- [ ] Add inline buttons
    - [ ] "View Dashboard" → dashboard URL
    - [ ] "Export CSV" → trigger export
- [ ] Test report delivery (all sections, images)

3.6 Streamlit Dashboard

- [ ] Create Streamlit app structure
    - [ ] `dashboard/app.py`
    - [ ] `dashboard/pages/` (6 pages)
    - [ ] `dashboard/utils/` (helpers)
- [ ] Implement Overview page
    - [ ] Date range selector
    - [ ] Key metrics cards
    - [ ] Compliance trend graph
    - [ ] Domain radar chart
    - [ ] Recent patterns list
- [ ] Implement Physical Health page
    - [ ] Sleep trend graph
    - [ ] Workout calendar heatmap
    - [ ] Body metrics input (manual)
- [ ] Implement Career Progress page
    - [ ] LeetCode progress graph
    - [ ] Job application funnel

    - [ ] Deep work hours trend
- [ ] Implement Mental Health page
    - [ ] Check-in completion calendar
    - [ ] Mood rating trend
    - [ ] Emotional session log
- [ ] Implement Wealth page
    - [ ] Net worth tracking graph
    - [ ] Monthly budget vs actual
    - [ ] SIP contribution tracking
- [ ] Implement Data Export page
    - [ ] Export check-ins (CSV)
    - [ ] Export metrics (CSV)
    - [ ] Download graphs (PNG)
- [ ] Add authentication (simple password for MVP)
- [ ] Test dashboard locally

3.7 Dashboard Deployment

- [ ] Update Dockerfile to include Streamlit
- [ ] Configure multi-port Cloud Run
    - [ ] Port 8000: FastAPI (bot)
    - [ ] Port 8501: Streamlit (dashboard)
- [ ] Deploy updated container
- [ ] Test dashboard access
- [ ] Optimize for mobile (responsive design)

3.8 Integration Testing

- [ ] Test weekly report generation end-to-end
- [ ] Test monthly report generation end-to-end
- [ ] Test scheduled triggers
- [ ] Test dashboard with real data
- [ ] Test all 6 dashboard pages
- [ ] Test export functions
- [ ] Verify graphs render correctly on mobile

Phase 3 Success Criteria:

- [ ] Weekly reports generated every Sunday 9 AM
- [ ] Monthly reports generated first Sunday of month
- [ ] All 4 graphs included in reports
- [ ] LLM summaries are actionable
- [ ] Dashboard accessible via URL
- [ ] All 6 pages functional
- [ ] Reports delivered to Telegram successfully

- [ ] Export functions work

PHASE 4: Advanced Features (Week 4)

Goal: Add emotional processing, ghosting detection, polish

Estimated Effort: 5-7 days
Status: Not Started

Tasks:

4.1 Emotional Processing Agent

- [ ] Create `EmotionalProcessingAgent` class
- [ ] Implement session state management
    - [ ] Multi-turn conversations
    - [ ] Session persistence
- [ ] Create keyword triggers
    - [ ] Detect: lonely, sad, porn, urge, ex, anxiety, etc.
    - [ ] Trigger emotional processing flow
- [ ] Implement CBT prompt templates
    - [ ] Cognitive restructuring template
    - [ ] Behavioral activation template
    - [ ] Urge surfing template
    - [ ] Reality testing template
- [ ] Add constitution grounding
    - [ ] Reference relevant constitution principles
    - [ ] Pull historical patterns (e.g., "last time you did X...")
- [ ] Implement session logging
    - [ ] Store in `emotional_sessions/{user_id}/sessions`
    - [ ] Track outcome (action committed, followed through)
- [ ] Add 24hr follow-up
    - [ ] Schedule follow-up message
    - [ ] "How did it go?"
    - [ ] Update session effectiveness
- [ ] Test emotional processing with various scenarios

4.2 Ghosting Detection

- [ ] Create `GhostingDetectionAgent` class
- [ ] Implement ghosting check logic
    - [ ] Count consecutive missed check-ins
    - [ ] Classify status: active/warning/critical/emergency

- [ ] Create escalation message templates
    - [ ] Day 2: Gentle reminder
    - [ ] Day 3: Urgent reminder
    - [ ] Day 4: Pattern analysis
    - [ ] Day 5: Emergency consideration
    - [ ] Day 7+: Daily reminders
- [ ] Implement return protocol
    - [ ] Welcome back message
    - [ ] Ask what happened
    - [ ] Restart without guilt
- [ ] Add Cloud Scheduler job (daily 10 PM)
    - [ ] Cron: `0 22 * * *`
    - [ ] Target: `/trigger/ghosting-check`
- [ ] Implement ghosting check endpoint
- [ ] Store ghosting events in Firestore
- [ ] Test ghosting detection with mock missed check-ins

4.3 Manual Commands

- [ ] Implement `/status` command
    - [ ] Show current streak
    - [ ] Show compliance score
    - [ ] Show current mode
- [ ] Implement `/mode` command
    - [ ] View current mode
    - [ ] Change mode (with confirmation)
- [ ] Implement `/weekly` command
    - [ ] Generate weekly report on-demand
- [ ] Implement `/monthly` command
    - [ ] Generate monthly review on-demand
- [ ] Implement `/export` command
    - [ ] Export check-ins as CSV
    - [ ] Send file via Telegram
- [ ] Implement `/dashboard` command
    - [ ] Send dashboard URL
- [ ] Test all commands

4.4 Error Handling & Resilience

- [ ] Add comprehensive error handling
    - [ ] Firestore connection errors
    - [ ] Vertex AI API errors
    - [ ] Telegram API errors
- [ ] Implement retry logic (exponential backoff)

- [ ] Add circuit breaker for external APIs
- [ ] Implement graceful degradation
  - [ ] If LLM fails, use template responses
  - [ ] If Firestore fails, log to Cloud Logging
- [ ] Add user-friendly error messages
  - [ ] "Something went wrong, trying again..."
  - [ ] "I'm having trouble right now, please try later"
- [ ] Test error scenarios

4.5 Logging & Monitoring

- [ ] Set up Cloud Logging
  - [ ] Structured logging (JSON)
  - [ ] Log levels: INFO, WARNING, ERROR
- [ ] Add key event logs
  - [ ] Check-in completed
  - [ ] Pattern detected
  - [ ] Intervention sent
  - [ ] Report generated
  - [ ] Error occurred
- [ ] Set up Cloud Monitoring
  - [ ] Create dashboard
  - [ ] Key metrics: response latency, error rate, check-in completion rate
- [ ] Set up alerts
  - [ ] Error rate >5%
  - [ ] Response latency >10s
  - [ ] Daily check-in completion <80%
- [ ] Test monitoring (trigger errors, check alerts)

4.6 Cost Optimization

- [ ] Audit Vertex AI usage
  - [ ] Token count per check-in
  - [ ] Optimize prompts (reduce tokens)
- [ ] Implement prompt caching
  - [ ] Cache constitution text
  - [ ] Cache common prompts
- [ ] Optimize Firestore queries
  - [ ] Use indexes
  - [ ] Limit query results
- [ ] Monitor GCP costs daily
- [ ] Document cost breakdown

4.7 Documentation

- [ ] Write README.md
    - [ ] Project overview
    - [ ] Architecture diagram
    - [ ] Setup instructions
    - [ ] Deployment guide
- [ ] Document API endpoints
- [ ] Document Firestore schema
- [ ] Document LangGraph agents
- [ ] Write troubleshooting guide
- [ ] Create user guide (for Ayush)
    - [ ] How to use the bot
    - [ ] Commands reference
    - [ ] Dashboard guide

4.8 Final Testing

- [ ] Full end-to-end test
    - [ ] Complete check-in
    - [ ] Trigger patterns
    - [ ] Receive interventions
    - [ ] Generate reports
    - [ ] Use emotional processing
    - [ ] Test ghosting detection
- [ ] Performance testing
    - [ ] Measure response times
    - [ ] Stress test (multiple concurrent users - if applicable)
- [ ] Security testing
    - [ ] Test authentication
    - [ ] Test webhook signature verification
- [ ] Usability testing (with Ayush)
    - [ ] Is the bot intuitive?
    - [ ] Are reports helpful?
    - [ ] Any pain points?

Phase 4 Success Criteria:

- [ ] Emotional processing agent works for all trigger types
- [ ] Ghosting detection escalates correctly
- [ ] All manual commands functional
- [ ] Comprehensive error handling in place
- [ ] Monitoring dashboard set up
- [ ] Cost <$5/month confirmed
- [ ] All documentation complete
- [ ] System passes end-to-end testing
- [ ] Ready for production use

---

8. TESTING STRATEGY

8.1 Unit Tests

Coverage Target: >80%

Test Files Structure:

```
tests/
├── test_agents/
│   ├── test_checkin_agent.py
│   ├── test_pattern_detection_agent.py
│   ├── test_report_agent.py
│   ├── test_emotional_processing_agent.py
│   └── test_intervention_agent.py
├── test_utils/
│   ├── test_compliance_score.py
│   ├── test_streak_counter.py
│   └── test_pattern_rules.py
├── test_services/
│   ├── test_firestore_service.py
│   └── test_llm_service.py
└── conftest.py (fixtures)
```

Key Test Cases:

```
# tests/test_utils/test_compliance_score.py
def test_compliance_score_all_complete():
    """Test compliance score when all tier 1 items completed"""
    responses = {
        'sleep': True,
        'training': True,
        'deep_work': True,
        'zero_porn': True,
        'boundaries': True
    }
    assert calculate_compliance_score(responses) == 100.0

def test_compliance_score_partial():
    """Test compliance score with partial completion"""
    responses = {
        'sleep': True,
        'training': False,
        'deep_work': True,
        'zero_porn': True,
        'boundaries': True
    }
    assert calculate_compliance_score(responses) == 80.0

# tests/test_agents/test_pattern_detection_agent.py
def test_detect_sleep_degradation():
    """Test sleep degradation pattern detection"""
    mock_data = [
        {'date': '2026-01-27', 'sleep_hours': 5.5},
        {'date': '2026-01-28', 'sleep_hours': 5.8},
        {'date': '2026-01-29', 'sleep_hours': 5.2}
    ]
    agent = PatternDetectionAgent()
    patterns = agent.detect_patterns(mock_data)
    assert len(patterns) == 1
    assert patterns[0]['pattern_type'] == 'sleep_degradation'
    assert patterns[0]['severity'] == 'warning'
```

8.2 Integration Tests

Test Scenarios:

```
# tests/integration/test_checkin_flow.py
@pytest.mark.integration
def test_full_checkin_flow(mock_telegram_bot, firestore_emulator):
    """Test complete check-in flow from start to finish"""
    # User sends /checkin
    response = send_telegram_command('/checkin', user_id='test_user')
    assert 'Time for your daily check-in' in response

    # Answer Question 1
    response = send_telegram_message('Sleep: Y, Training: Y, Deep Work: Y, Zero Porn: Y,
Boundaries: Y')
    assert 'Question 2' in response

    # Answer Question 2
    response = send_telegram_message('No major challenges today')
    assert 'Question 3' in response
```

```
# Answer Question 3
response = send_telegram_message('8 - solid day overall')
assert 'Question 4' in response

# Answer Question 4
response = send_telegram_message('Priority: LeetCode. Obstacle: Meeting might run late.')
assert 'Check-in complete' in response
assert 'Streak' in response

# Verify data in Firestore
checkin = firestore_emulator.get_checkin('test_user', '2026-01-29')
assert checkin is not None
assert checkin['compliance_score'] == 100.0
```

8.3 End-to-End Tests

Test Scenarios:

1. **Scenario: New user onboarding**
    - User starts bot with `/start`
    - Bot creates user profile
    - User completes first check-in
    - Streak initializes to 1
    - Verify Firestore data
2. **Scenario: Pattern detection triggers intervention**
    - User completes 3 check-ins with <6 hrs sleep
    - Pattern detection runs
    - Sleep degradation pattern detected
    - Warning intervention sent
    - User acknowledges
    - Verify intervention logged
3. **Scenario: Weekly report generation**
    - 7 days of check-in data exists
    - Cloud Scheduler triggers report
    - Report agent aggregates metrics
    - Graphs generated
    - Report sent to user
    - Verify report stored

8.4 Manual Testing Checklist

Pre-Production Testing:

Check-In Flow:

- [ ] Complete check-in with all "Yes" responses
- [ ] Complete check-in with mixed responses
- [ ] Start check-in, pause, resume later
- [ ] Start check-in, timeout, verify reminder sent
- [ ] Complete multiple check-ins over several days
- [ ] Verify streak increments correctly
- [ ] Verify streak resets after 48hr gap

Pattern Detection:

- [ ] Create 3 days of <6hr sleep data
- [ ] Verify sleep degradation pattern detected
- [ ] Verify warning intervention sent
- [ ] Create 3 days of missed workouts
- [ ] Verify training abandonment detected
- [ ] Create porn relapse data (3 in a week)
- [ ] Verify intervention sent

Reports:

- [ ] Trigger weekly report manually with `/weekly`
- [ ] Verify all sections present
- [ ] Verify 4 graphs attached
- [ ] Verify LLM summary is relevant
- [ ] Trigger monthly report manually
- [ ] Verify goal progress section
- [ ] Verify deep pattern analysis

Emotional Processing:

- [ ] Send "I'm feeling lonely" message
- [ ] Verify emotional processing triggered
- [ ] Complete full session
- [ ] Verify session logged
- [ ] Verify 24hr follow-up scheduled

Ghosting Detection:

- [ ] Miss 2 check-ins
- [ ] Verify gentle reminder sent
- [ ] Miss 3 check-ins
- [ ] Verify urgent reminder sent

- [ ] Return after ghosting
- [ ] Verify welcome back message

Dashboard:

- [ ] Access dashboard URL
- [ ] Test all 6 pages
- [ ] Verify graphs render correctly
- [ ] Test export functions
- [ ] Test on mobile device

Commands:

- [ ] Test `/start`
- [ ] Test `/help`
- [ ] Test `/status`
- [ ] Test `/mode`
- [ ] Test `/weekly`
- [ ] Test `/monthly`
- [ ] Test `/export`
- [ ] Test `/dashboard`

---

9. DEPLOYMENT STRATEGY

9.1 GCP Project Setup

Checklist:

- [ ] Create GCP project: `constitution-agent-prod`
- [ ] Enable billing
- [ ] Enable APIs:
    - [ ] Cloud Run API
    - [ ] Firestore API (Native mode)
    - [ ] Vertex AI API
    - [ ] Cloud Scheduler API
    - [ ] Cloud Storage API
    - [ ] Cloud Logging API
    - [ ] Cloud Monitoring API
    - [ ] Secret Manager API
- [ ] Set up IAM roles:
    - [ ] Cloud Run Invoker (for Cloud Scheduler)

    - [ ] Firestore User (for app)
    - [ ] Vertex AI User (for app)
    - [ ] Storage Object Admin (for graphs)
- [ ] Create service account for application
- [ ] Download service account key (store securely)
- [ ] Add key to Secret Manager

9.2 Firestore Setup

Checklist:

- [ ] Create Firestore database
    - [ ] Mode: Native
    - [ ] Location: asia-south1 (Mumbai)
- [ ] Create collections:
    - [ ] `users`
    - [ ] `daily_checkins/{user_id}/checkins`
    - [ ] `metrics/{user_id}/weekly`
    - [ ] `metrics/{user_id}/monthly`
    - [ ] `patterns/{user_id}/detected`
    - [ ] `interventions/{user_id}/history`
    - [ ] `emotional_sessions/{user_id}/sessions`
- [ ] Create indexes:
    - [ ] `daily_checkins`: (user_id, date)
    - [ ] `patterns`: (user_id, detected_at, severity)
    - [ ] `interventions`: (user_id, triggered_at)
- [ ] Set security rules (authenticated access only)

9.3 Cloud Storage Setup

Checklist:

- [ ] Create bucket: `constitution-agent-graphs`
    - [ ] Location: asia-south1
    - [ ] Storage class: Standard
- [ ] Set lifecycle policy (delete files >90 days old)
- [ ] Set IAM permissions (app can write)

9.4 Telegram Bot Setup

Checklist:

- [ ] Create bot via @BotFather

    - [ ] Command: `/newbot`
    - [ ] Name: "Constitution Agent"
    - [ ] Username: `@constitution_ayush_bot`
- [ ] Get bot token
- [ ] Store token in Secret Manager

[ ] Set bot commands via BotFather:
 start - Initialize botcheckin - Daily check-instatus - View streak & complianceweekly - Weekly reportmonthly - Monthly reviewhelp - Show commandsmode - View/change modedashboard - Dashboard URLexport - Export datasettings - Preferences

-
- [ ] Set bot description
- [ ] Set bot profile picture (optional)

9.5 Cloud Run Deployment

Dockerfile:

```
FROM python:3.11-slim

WORKDIR /app

# Install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Expose ports
EXPOSE 8000 8501

# Start both FastAPI and Streamlit
CMD ["sh", "-c", "uvicorn main:app --host 0.0.0.0 --port 8000 & streamlit run dashboard/app.py --server.port 8501 --server.address 0.0.0.0"]
```

Deployment Commands:

```
# Build image
gcloud builds submit --tag gcr.io/constitution-agent-prod/constitution-agent

# Deploy to Cloud Run
gcloud run deploy constitution-agent \
  --image gcr.io/constitution-agent-prod/constitution-agent \
  --platform managed \
  --region asia-south1 \
  --memory 512Mi \
  --cpu 1 \
  --min-instances 0 \
  --max-instances 3 \
  --timeout 300 \
  --allow-unauthenticated \
  --set-env-vars GCP_PROJECT=constitution-agent-prod \
  --set-secrets TELEGRAM_BOT_TOKEN=telegram-bot-token:latest
```

Post-Deployment:

- [ ] Get Cloud Run URL

[ ] Set Telegram webhook:

```
 curl
https://api.telegram.org/bot<TOKEN>/setWebhook?url=<CLOUD_RUN_URL>/webhook/telegram
```

-
- [ ] Test webhook: send `/start` to bot
- [ ] Verify logs in Cloud Logging

9.6 Cloud Scheduler Setup

Create Jobs:

```
# Daily check-in trigger (9 PM IST)
gcloud scheduler jobs create http daily-checkin-trigger \
  --schedule="0 21 * * *" \
  --uri="https://<CLOUD_RUN_URL>/trigger/daily-checkin" \
  --http-method=POST \
  --time-zone="Asia/Kolkata"

# Weekly report trigger (Sunday 9 AM IST)
gcloud scheduler jobs create http weekly-report-trigger \
  --schedule="0 9 * * 0" \
  --uri="https://<CLOUD_RUN_URL>/trigger/weekly-report" \
  --http-method=POST \
  --time-zone="Asia/Kolkata"

# Monthly review trigger (1st Sunday 9 AM IST)
```

```
gcloud scheduler jobs create http monthly-review-trigger \
  --schedule="0 9 1-7 * 0" \
  --uri="https://<CLOUD_RUN_URL>/trigger/monthly-review" \
  --http-method=POST \
  --time-zone="Asia/Kolkata"

# Pattern scan trigger (every 6 hours)
gcloud scheduler jobs create http pattern-scan-trigger \
  --schedule="0 */6 * * *" \
  --uri="https://<CLOUD_RUN_URL>/trigger/pattern-scan" \
  --http-method=POST \
  --time-zone="Asia/Kolkata"

# Ghosting check trigger (daily 10 PM IST)
gcloud scheduler jobs create http ghosting-check-trigger \
  --schedule="0 22 * * *" \
  --uri="https://<CLOUD_RUN_URL>/trigger/ghosting-check" \
  --http-method=POST \
  --time-zone="Asia/Kolkata"
```

Test Jobs:

- [ ] Manually trigger each job
- [ ] Verify logs show job executed
- [ ] Verify expected outcome (check-in sent, report generated, etc.)

9.7 Monitoring Setup

Cloud Monitoring Dashboard:

- [ ] Create dashboard: "Constitution Agent - Production"
- [ ] Add charts:
    - [ ] Request latency (95th percentile)
    - [ ] Error rate (%)
    - [ ] Check-in completion rate (daily)
    - [ ] Pattern detection count (daily)
    - [ ] Intervention count (daily)
    - [ ] Vertex AI API calls (daily)
    - [ ] Cloud Run CPU utilization
    - [ ] Cloud Run memory utilization

Alerts:

```
# Create alert policy for high error rate
```

```
gcloud alpha monitoring policies create \
  --notification-channels=<CHANNEL_ID> \
  --display-name="High Error Rate" \
  --condition-display-name="Error rate > 5%" \
  --condition-threshold-value=5 \
  --condition-threshold-duration=300s
```

- [ ] Alert: Error rate >5%
- [ ] Alert: Response latency >10s
- [ ] Alert: Check-in completion <80%
- [ ] Alert: Cloud Run instance down

Notification Channels:

- [ ] Email: ayush@example.com
- [ ] Telegram: Send alert to bot channel (optional)

9.8 Rollback Plan

In case of critical bugs:

Immediate: Rollback to previous Cloud Run revision

```
 gcloud run services update-traffic constitution-agent \
  --to-revisions=<PREVIOUS_REVISION>=100
```

1.

Pause Scheduled Jobs:

```
 gcloud scheduler jobs pause daily-checkin-trigger
gcloud scheduler jobs pause weekly-report-trigger
# ... pause all jobs
```

2.
3. **Fix Bug:**
   - Fix code locally
   - Test thoroughly
   - Redeploy
4. **Resume:**
   - Resume scheduled jobs
   - Monitor closely

---

10. MONITORING & MAINTENANCE

10.1 Key Metrics to Monitor

User Engagement:

- Check-in completion rate (daily, weekly, monthly)
- Streak distribution (how many users at 7, 30, 90 days?)
- Active users (daily, weekly)
- Ghosting rate (% of users who ghost >3 days)
- Return rate (% of ghosted users who return)

System Performance:

- Response latency (p50, p95, p99)
- Error rate (%)
- API call count (Vertex AI, Firestore, Telegram)
- Cloud Run instance count
- Memory/CPU utilization

AI Quality:

- Pattern detection accuracy (% of true positives)
- Intervention effectiveness (% leading to behavior change)
- LLM response quality (manual review)
- Report usefulness (user feedback)

Cost:

- Daily cost breakdown (by service)
- Monthly total cost
- Cost per check-in
- Cost per report

10.2 Daily Monitoring Tasks

Checklist (5 min/day):

- [ ] Check error logs (any critical errors?)
- [ ] Check check-in completion rate (>80%?)
- [ ] Check Cloud Run status (all healthy?)
- [ ] Check GCP costs (under $0.20/day?)
- [ ] Check user messages (any feedback/complaints?)

Dashboard URL: [Link to Cloud Monitoring Dashboard]

10.3 Weekly Maintenance Tasks

Checklist (15 min/week):

- [ ] Review pattern detection accuracy (sample 10 patterns)
- [ ] Review intervention effectiveness (sample 5 interventions)
- [ ] Review LLM response quality (sample 10 check-in feedbacks)
- [ ] Check for outdated dependencies (security updates)
- [ ] Review user feedback (Telegram messages)
- [ ] Check Firestore size (delete old data if >1GB)
- [ ] Review cost trends (any spikes?)

10.4 Monthly Maintenance Tasks

Checklist (30 min/month):

- [ ] Review all metrics (engagement, performance, cost)
- [ ] Analyze user behavior trends (what's working? what's not?)
- [ ] Review constitution effectiveness (is it helping user?)
- [ ] Plan improvements (new features, optimizations)
- [ ] Update documentation (if code changed)
- [ ] Backup critical data (export Firestore data)
- [ ] Security audit (check IAM roles, secrets)
- [ ] Dependency updates (upgrade to latest stable versions)

10.5 Incident Response Plan

Severity Levels:

P0 (Critical - Immediate Response):

- Bot completely down (doesn't respond)
- Data loss
- Security breach

Response: Drop everything, fix immediately, rollback if needed

P1 (High - 1hr Response):

- High error rate (>20%)
- Scheduled jobs failing
- LLM not responding

Response: Investigate within 1hr, fix within 4hrs

P2 (Medium - 4hr Response):

- Moderate error rate (5-20%)
- Slow responses (>10s)
- Non-critical bugs

Response: Investigate within 4hrs, fix within 24hrs

P3 (Low - Best Effort):

- Minor UI issues
- Feature requests
- Optimizations

Response: Add to backlog, fix when time permits

Incident Log:

- [ ] Create `INCIDENTS.md` file

[ ] Template:
## Incident YYYY-MM-DD: [Title]**Severity:** P0/P1/P2/P3**Detected At:** YYYY-MM-DD HH:MM IST**Resolved At:** YYYY-MM-DD HH:MM IST**Root Cause:** [Description]**Impact:** [Users affected, duration]**Resolution:** [What was done]**Prevention:** [How to prevent in future]

-

---

11. COST ANALYSIS

11.1 Cost Breakdown (Projected)

Monthly Cost Estimate (Single User):

Service	Usage	Unit Cost	Monthly Cost
Vertex AI (Gemini 2.0 Flash)	100K tokens/month	$0.0005/1K tokens	$0.05
Cloud Run	<2M requests, <10GB-seconds	Free tier	$0.00

Firestore	50K reads, 10K writes, 1GB storage	Free tier	$0.00
Cloud Storage	<1GB storage, <10K ops	Free tier	$0.00
Cloud Scheduler	5 jobs	$0.10/job/month	$0.50
Cloud Logging	<50GB logs	Free tier (first 50GB)	$0.00
Cloud Monitoring	Basic metrics	Free	$0.00
Telegram Bot API	Unlimited messages	Free	$0.00
TOTAL			**$0.55/month** ✅

Notes:

- All estimates assume single user (Ayush)
- Free tier limits are generous for this use case
- Biggest cost: Cloud Scheduler ($0.50/month for 5 jobs)
- Vertex AI cost negligible with Gemini Flash (~$0.05/month)
- Well under $5/month budget

Cost Optimization Tips:

1. Use Gemini 2.0 Flash (cheapest model)
2. Optimize prompts to minimize tokens
3. Cache repeated prompts
4. Use free tier for all GCP services where possible
5. Delete old data (keep last 90 days only)

11.2 Cost Monitoring

Set up Budget Alerts:

```
# Create budget
gcloud billing budgets create \
  --billing-account=<BILLING_ACCOUNT_ID> \
  --display-name="Constitution Agent Budget" \
  --budget-amount=5.00 \
  --threshold-rules-percent=50,80,100
```

Alerts:

- [ ] 50% of budget ($2.50) → Warning email
- [ ] 80% of budget ($4.00) → Urgent email
- [ ] 100% of budget ($5.00) → Critical email + investigate

Cost Dashboard:

- Daily cost tracking spreadsheet
- Weekly cost review
- Identify any cost spikes immediately

---

12. RISK MITIGATION

12.1 Technical Risks

Risk 1: Firestore Free Tier Exceeded

- **Likelihood:** Low
- **Impact:** Medium (extra cost)
- **Mitigation:**
  - Monitor Firestore usage daily
  - Set up alert at 80% of free tier
  - Delete old data (keep last 90 days)
  - Optimize queries (use indexes)

Risk 2: Vertex AI Cost Spike

- **Likelihood:** Low
- **Impact:** High (could blow budget)
- **Mitigation:**
  - Use Gemini Flash (cheapest model)
  - Optimize prompts (reduce tokens)
  - Cache repeated prompts
  - Set up cost alert at $1/day
  - Implement rate limiting if needed

Risk 3: Cloud Run Cold Starts

- **Likelihood:** Medium (with min instances = 0)
- **Impact:** Low (slow first response)
- **Mitigation:**
  - Accept cold starts (user won't notice for most requests)
  - For critical scheduled jobs, set min-instances=1 temporarily
  - Optimize container size (reduce cold start time)

Risk 4: Telegram Bot API Rate Limits

- **Likelihood:** Low (for single user)
- **Impact:** Medium (messages fail)
- **Mitigation:**
    - Respect Telegram rate limits (30 messages/sec)
    - Implement retry logic with exponential backoff
    - Queue messages if burst needed

Risk 5: Data Loss (Firestore)

- **Likelihood:** Very Low (GCP is reliable)
- **Impact:** High (lose all data)
- **Mitigation:**
    - Firestore has automatic backups
    - Export data monthly to Cloud Storage (CSV)
    - Test restore procedure

12.2 User Experience Risks

Risk 6: Bot Feels Robotic/Impersonal

- **Likelihood:** Medium
- **Impact:** Medium (user disengagement)
- **Mitigation:**
    - Invest in prompt engineering (supportive but firm tone)
    - Use constitution grounding (reference personal context)
    - Test responses with Ayush, iterate on tone
    - Add personality to messages (emoji, encouragement)

Risk 7: User Finds Check-In Tedious

- **Likelihood:** Medium
- **Impact:** High (abandonment)
- **Mitigation:**
    - Keep check-in <3 minutes
    - Add inline keyboards (Y/N buttons)
    - Allow voice messages (future feature)
    - Make feedback valuable (not generic)

Risk 8: Reports Not Actionable

- **Likelihood:** Medium
- **Impact:** Medium (wasted effort)
- **Mitigation:**
    - Focus on 2-3 specific actions per report

    - Use constitution language (reference principles)
    - Test with Ayush, get feedback
    - Iterate on LLM prompts for better insights

Risk 9: Pattern Detection False Positives

- **Likelihood:** Medium
- **Impact:** Low (annoying messages)
- **Mitigation:**
    - Set conservative thresholds (e.g., 3 violations, not 2)
    - Allow user to dismiss false positives
    - Track false positive rate, adjust rules
    - Let user configure sensitivity (future feature)

12.3 Privacy & Security Risks

Risk 10: Unauthorized Access to Dashboard

- **Likelihood:** Low
- **Impact:** High (privacy breach)
- **Mitigation:**
    - Implement authentication (password + Google SSO)
    - Use HTTPS only
    - Session timeout after 1 hour
    - Log all access attempts

Risk 11: Telegram Account Compromise

- **Likelihood:** Low
- **Impact:** High (attacker sees all data)
- **Mitigation:**
    - Ayush uses strong Telegram password
    - Enable 2FA on Telegram
    - Bot verifies user ID (not just username)
    - Add emergency "lock account" command

Risk 12: Data Breach (GCP Compromise)

- **Likelihood:** Very Low
- **Impact:** High
- **Mitigation:**
    - Use GCP IAM best practices (least privilege)
    - Rotate service account keys regularly
    - Enable Cloud Audit Logs
    - Monitor for suspicious activity

---

13. SUCCESS METRICS (30/90/180 Days)

30-Day Success Metrics (Target: Feb 27, 2026)

User Engagement:

- [ ] Check-in completion rate >85% (25/30 days)
- [ ] Zero ghosting events (missed <3 days)
- [ ] Streak maintained (30+ days if started on time)

System Performance:

- [ ] Response latency <5 seconds (95th percentile)
- [ ] Error rate <2%
- [ ] Uptime >99%

AI Quality:

- [ ] Pattern detection caught 100% of critical violations
- [ ] Intervention effectiveness >60% (behavior corrected)
- [ ] LLM feedback rated "helpful" by user (manual review)

Cost:

- [ ] Monthly cost <$1.00 (well under budget)

90-Day Success Metrics (Target: Apr 27, 2026)

User Engagement:

- [ ] Check-in completion rate >90% (81/90 days)
- [ ] Longest streak >60 days
- [ ] User actively uses emotional processing (5+ sessions)
- [ ] User actively uses dashboard (views weekly)

System Performance:

- [ ] All metrics maintained from 30-day
- [ ] Zero critical incidents (P0)
- [ ] <3 high-priority incidents (P1)

AI Quality:

- [ ] Pattern detection accuracy >85%

- [ ] Intervention effectiveness >70%
- [ ] User reports "constitution is working" (subjective feedback)

Cost:

- [ ] Monthly cost <$2.00 (still well under budget)

Bonus:

- [ ] User completes surgery recovery successfully (constitution supported him)
- [ ] User starts job search with momentum (career prep on track)

180-Day Success Metrics (Target: July 27, 2026)

User Engagement:

- [ ] Check-in completion rate >90% (162/180 days)
- [ ] Longest streak >90 days
- [ ] User views reports regularly, finds them valuable
- [ ] User recommends system to friends (word of mouth)

System Performance:

- [ ] All metrics maintained
- [ ] System considered "production-grade" (reliable, fast, bug-free)

AI Quality:

- [ ] Pattern detection fine-tuned based on 180 days of data
- [ ] Intervention system adapted to user's patterns
- [ ] Reports provide genuinely useful insights (not generic)

Impact:

- [ ] User's constitution compliance at all-time high
- [ ] User reports life improvements (physique, career, mental health)
- [ ] User hit ₹2L/month income threshold (career goal)
- [ ] User dating successfully (relationship goal on track)

Cost:

- [ ] Monthly cost still <$3.00 (sustainable long-term)

Future:

- [ ] System ready to scale to multiple users (if desired)
- [ ] Architecture documented for future enhancements

APPENDIX A: GLOSSARY

Compliance Score: Percentage of Tier 1 non-negotiables completed (0-100%)

Constitution Mode: User's current life phase (optimization, maintenance, survival)

Ghosting: Missing 3+ consecutive daily check-ins

Intervention Level: Severity of AI response to pattern violation (nudge, warning, veto)

Pattern: Detected behavioral trend that violates constitution rules

Streak: Consecutive days of check-in completion

Tier 1 Non-Negotiables: 5 daily must-dos (sleep, training, deep work, zero porn, boundaries)

APPENDIX B: CONTACT & SUPPORT

Project Owner: Ayush
Email: ayush@example.com
Telegram: @ayush_username

GCP Project: constitution-agent-prod
Cloud Run URL: https://constitution-agent-<hash>-uc.a.run.app
Dashboard URL: https://constitution-agent-<hash>-uc.a.run.app:8501
Telegram Bot: @constitution_ayush_bot

Documentation: [GitHub repo README]
Monitoring Dashboard: [Cloud Monitoring dashboard URL]
Cost Dashboard: [GCP Billing dashboard URL]

APPENDIX C: NEXT STEPS (After MVP)

Future Features (Post-Phase 4):

Priority 1 (Next 3 months):

1. Voice message support (check-in via voice)

2. Photo tracking (body progress pics, meal logging)
3. Habit streaks visualization (calendar heatmap)
4. Accountability partner integration (emergency contact)
5. Custom pattern rules (user can add own patterns)

Priority 2 (Next 6 months):

1. Multi-user support (scale to friends/family)
2. Social features (compete with friends on streaks)
3. Integration with fitness apps (sync workouts, sleep)
4. Advanced analytics (correlation analysis: sleep vs mood, etc.)
5. Custom report scheduling

Priority 3 (Future):

1. Mobile app (native iOS/Android)
2. Wearable integration (Apple Watch, Fitbit)
3. AI coach (proactive suggestions, not just reactive)
4. Gamification (badges, levels, achievements)
5. Export to notion/obsidian

END OF PRD

Document Version: 1.0
Last Updated: January 29, 2026
Status: Ready for Implementation

This PRD is comprehensive and actionable. All tasks are broken down with checklists. Ready to start Phase 1?