

Agentic Honeypot API: Complete Build Plan & PRD

Product: SentinelHP - Agentic Honeypot for Scam Detection & Intelligence Extraction

Hackathon: GUVI/HCL AI for Impact - AI for Fraud Detection & User Safety

Timeline: 2 Days with Integrated Testing

Prepared: February 2026

Executive Summary

This document provides a comprehensive plan to build and deploy an Agentic Honeypot API system within 2 days. The system detects scam messages, autonomously engages scammers using AI agents, extracts actionable intelligence (UPI IDs, bank accounts, phishing links), and reports structured results to an evaluation endpoint.

The plan integrates development, testing, and API endpoint validation as required by organizers.

Part 1: Simplified 2-Day Build Plan

Day 1: Core Infrastructure (8-10 hours)

Phase 1A: API Foundation (2-3 hours) - UPDATED FOR GUVI FORMAT

Objective: Working API endpoint with GUVI-compliant schema

Tasks:

1. Set up FastAPI project structure
2. Implement API key authentication middleware (header: x-api-key)
3. Create POST endpoint (e.g., /api/honeytrap or /api/message)
4. Add request schema validation for GUVI format
5. Implement basic logging

Request Schema (Pydantic Models):

```
class Message(BaseModel):
```

```
    sender: str # "scammer" or "user"
```

```
    text: str
```

```
    timestamp: str
```

```
class ConversationMessage(BaseModel):
```

```
    sender: str
```

```
    text: str
```

```
    timestamp: str
```

```
class Metadata(BaseModel):
    channel: str
    language: str
    locale: str

class HoneyPotRequest(BaseModel):
    sessionId: str
    message: Message
    conversationHistory: List[ConversationMessage] = []
    metadata: Optional[Metadata] = None

class HoneyPotResponse(BaseModel):
    status: str = "success"
    reply: str
```

Deliverables:

- Working API accepting GUVI-format requests
- API key validation functional (x-api-key header)
- Response format matching GUVI specs

Testing Checkpoint:

```
curl -X POST https://your-endpoint/api/honeypot
-H "x-api-key: your-secret-key"
-H "Content-Type: application/json"
-d '{
  "sessionId": "test-session-001",
  "message": {
    "sender": "scammer",
    "text": "Your bank account will be blocked today. Verify immediately.",
    "timestamp": "2026-02-02T10:15:30Z"
  },
  "conversationHistory": [],
  "metadata": {
    "channel": "SMS",
    "language": "English",
    "locale": "IN"
  }
}'
```

Expected response:

```
{
  "status": "success",
  "reply": "Why will my account be blocked?"
}
```

Phase 1B: Database Setup (1-2 hours)

Objective: Session persistence with PostgreSQL

Schema Design:

-- Sessions table

```
CREATE TABLE sessions (  
  session_id VARCHAR(255) PRIMARY KEY,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  status VARCHAR(50),  
  risk_score FLOAT,  
  agent_engaged BOOLEAN DEFAULT FALSE  
);
```

-- Messages table

```
CREATE TABLE messages (  
  id SERIAL PRIMARY KEY,  
  session_id VARCHAR(255) REFERENCES sessions(session_id),  
  turn_number INTEGER,  
  sender VARCHAR(50),  
  message TEXT,  
  timestamp TIMESTAMP,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

-- Extracted entities table

```
CREATE TABLE extracted_entities (  
  id SERIAL PRIMARY KEY,  
  session_id VARCHAR(255) REFERENCES sessions(session_id),  
  entity_type VARCHAR(50),  
  entity_value TEXT,  
  confidence FLOAT,  
  source_turn INTEGER,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

Tasks:

1. Set up PostgreSQL (or SQLite for quick demo)
2. Create database schema
3. Implement SQLAlchemy models
4. Add session management functions

Testing Checkpoint:

- Verify session creation in database
 - Test message logging
 - Confirm entity insertion works
-

Phase 1C: Scam Detection Layer (3-4 hours)

Objective: Layered detection system (rules → LLM)

Rule-Based Heuristics:

```
SCAM_KEYWORDS = {  
'urgency': ['urgent', 'immediately', 'now', 'today', 'blocked', 'suspended'],  
'authority': ['bank', 'police', 'government', 'RBI', 'income tax', 'courier'],  
'financial': ['refund', 'cashback', 'prize', 'lottery', 'verify account', 'KYC'],  
'credentials': ['OTP', 'PIN', 'password', 'CVV', 'card number', 'Aadhaar']  
}
```

```
ENTITY_PATTERNS = {  
'upi': r'\b[\w.-]+@[\w]+\b',  
'phone': r'+91[\d]{10}|\b[6-9]\d{9}\b',  
'url': r'https?://[^\s]+',  
'bank_account': r'\b\d{9,18}\b'  
}
```

Detection Pipeline:

1. **Rule Layer:** Score based on keyword presence, entity detection, urgency patterns
2. **Behavioral Layer:** Unknown sender + financial request + urgency = high risk
3. **LLM Fallback:** Google Gemini classification for ambiguous cases

Output Structure:

```
{  
  "risk_score": 0.87,  
  "confidence": "high",  
  "signals": ["urgency_detected", "upi_present", "authority_impersonation"],  
  "should_engage": true  
}
```

Testing Checkpoint:

```
test_messages = [  
  "Your UPI will be blocked. Verify now: fraud@paytm",  
  "Hello, how are you?",  
  "URGENT: KYC update needed. Call +919876543210"  
]
```

Verify correct risk scores and signal detection

Phase 1D: Google ADK Agent Setup (2 hours)

Objective: Working AI agent with persona

System Prompt Template:

You are a human user engaging with someone via SMS/WhatsApp.

PERSONA: {persona_description}

STRICT RULES:

1. Never admit you are an AI
2. Never provide real credentials, bank details, or OTPs
3. Ask clarifying questions when confused
4. Show mild hesitation and request repetition
5. Mirror the language style of the sender
6. Never confront or accuse anyone of scamming

CONVERSATION GOAL:

Understand what the sender wants while appearing cooperative but slightly confused.

MEMORY:

{conversation_history}

Respond naturally in 1-2 sentences.

Persona Examples:

- **Digitally Naive:** "I'm not very good with technology. Which app should I open exactly?"
- **Average User:** "I tried but it shows pending. What should I do next?"
- **Skeptical:** "This seems unusual. Can you confirm you're from the bank?"

Testing Checkpoint:

- Send test message to agent
- Verify human-like response
- Confirm no AI disclosure

Day 2: Intelligence & Integration (8-10 hours)

Phase 2A: Intelligence Extraction Pipeline (2-3 hours)

Objective: High-confidence entity extraction

Extraction Logic:

```
class IntelligenceExtractor:
def extract_entities(self, message: str, conversation_history: list) -> dict:
    entities = {
    'upi_ids': [],
    'bank_accounts': [],
    'phone_numbers': [],
```

```
'urls': []
}
```

```
# Regex extraction
entities['upi_ids'] = self.validate_upi(re.findall(UPI_PATTERN, message))
entities['phone_numbers'] = self.validate_phone(re.findall(PHONE_PATTERN,
entities['urls'] = self.validate_url(re.findall(URL_PATTERN, message))

# Contextual confirmation using LLM
confirmed = self.llm_confirm_entities(entities, conversation_history)

# Only return high-confidence (>0.85)
return self.filter_by_confidence(confirmed, threshold=0.85)
```

Validation Rules:

Entity Type	Validation
UPI ID	Format: user@provider, provider in known list
Phone	+91 prefix or 10-digit starting with 6-9
Bank Account	9-18 digits, checksum if IFSC present
URL	Valid domain, not whitelisted legitimate sites

Table 1: Entity Validation Criteria

Testing Checkpoint:

```
test_message = "Send payment to fraud123@paytm. Call me at +919876543210. Visit
http://fake-bank.site"
extracted = extractor.extract_entities(test_message, [])
assert len(extracted['upi_ids']) == 1
assert len(extracted['phone_numbers']) == 1
assert extracted['upi_ids'][0]['confidence'] > 0.85
```

Phase 2B: Multi-Turn Conversation Loop (2 hours)

Objective: Sustained agent engagement with memory

Conversation Controller:

```
class ConversationController:
    MAX_TURNS = 10
    TIMEOUT_SECONDS = 300
```

```

async def handle_turn(self, session_id: str, message: str):
    session = self.db.get_session(session_id)

    # Check termination conditions
    if self.should_terminate(session):
        await self.finalize_session(session_id)
        return

    # Load conversation history
    history = self.db.get_messages(session_id)

    # Generate agent response
    response = await self.agent.generate_response(
        message=message,
        history=history,
        persona=session.persona
    )

    # Extract intelligence from current turn
    entities = self.extractor.extract_entities(message, history)

    # Store turn
    self.db.save_message(session_id, message, response)
    self.db.save_entities(session_id, entities)

    return response

def should_terminate(self, session) -> bool:
    return (
        session.turn_count >= self.MAX_TURNS or
        session.duration > self.TIMEOUT_SECONDS or
        session.intelligence_confidence >= 0.9 or
        session.safety_override
    )

```

Testing Checkpoint:

- Simulate 5-turn conversation

- Verify memory persistence
- Confirm termination conditions work

Phase 2C: Evaluation Callback Implementation (1-2 hours) - GUVI COMPLIANT

Objective: Reliable callback to GUVI evaluation endpoint

Callback Endpoint: <https://hackathon.guvi.in/api/updateHoneyPotFinalResult>

EXACT Callback Payload Structure (CRITICAL - Must match GUVI spec):

```
{
  "sessionId": "abc123-session-id",
  "scamDetected": true,
  "totalMessagesExchanged": 18,
  "extractedIntelligence": {
    "bankAccounts": ["XXXX-XXXX-XXXX", "9876543210"],
    "upiIds": ["scammer@upi", "fraud@paytm"],
    "phishingLinks": ["http://malicious-link.example", "http://fake-bank.site"],
    "phoneNumbers": ["+91XXXXXXXXXX", "+919876543210"],
    "suspiciousKeywords": ["urgent", "verify now", "account blocked", "KYC update"]
  },
  "agentNotes": "Scammer used urgency tactics and payment redirection. Impersonated bank authority."
}
```

Implementation Example:

```
class EvaluationCallback:
    GUVI_ENDPOINT = "https://hackathon.guvi.in/api/updateHoneyPotFinalResult"
    MAX_RETRIES = 3
    RETRY_DELAY = [1, 3, 5]
```

```
    async def send_callback(self, session_id: str):
        # Build GUVI-compliant payload
        payload = self.build_guvi_payload(session_id)

        for attempt in range(self.MAX_RETRIES):
            try:
                response = await self.http_client.post(
                    url=self.GUVI_ENDPOINT,
                    json=payload,
                    headers={"Content-Type": "application/json"},
                    timeout=10
                )
```



```

        if response.status_code == 200:
            self.db.mark_callback_sent(session_id)
            logger.info(f"GUVI callback successful: {session_id}")
            return

    except Exception as e:
        logger.error(f"GUVI callback attempt {attempt+1} failed: {e}")
        await asyncio.sleep(self.RETRY_DELAY[attempt])

    self.db.mark_callback_failed(session_id)

def build_guvi_payload(self, session_id: str) -> dict:
    session = self.db.get_session(session_id)
    entities = self.db.get_entities(session_id)
    messages = self.db.get_messages(session_id)

    return {
        "sessionId": session_id,
        "scamDetected": session.scam_detected,
        "totalMessagesExchanged": len(messages),
        "extractedIntelligence": {
            "bankAccounts": [e['value'] for e in entities if e['type'] == 'bank_account'],
            "upiIds": [e['value'] for e in entities if e['type'] == 'upi_id'],
            "phishingLinks": [e['value'] for e in entities if e['type'] == 'url'],
            "phoneNumbers": [e['value'] for e in entities if e['type'] == 'phone'],
            "suspiciousKeywords": self.extract_keywords(messages)
        },
        "agentNotes": self.generate_agent_notes(session, messages)
    }

```

Retry Logic:

```

class EvaluationCallback:
    MAX_RETRIES = 3
    RETRY_DELAY = [1, 3, 5] # seconds

```

```

async def send_callback(self, session_id: str):
    payload = self.build_payload(session_id)

```

```

for attempt in range(self.MAX_RETRIES):
    try:
        response = await self.http_client.post(
            url=EVALUATION_ENDPOINT,
            json=payload,
            headers={"X-API-Key": EVALUATION_KEY},
            timeout=10
        )

        if response.status_code == 200:
            self.db.mark_callback_sent(session_id)
            logger.info(f"Callback successful: {session_id}")
            return

    except Exception as e:
        logger.error(f"Callback attempt {attempt+1} failed: {e}")
        await asyncio.sleep(self.RETRY_DELAY[attempt])

self.db.mark_callback_failed(session_id)

```

Testing Checkpoint:

- Mock evaluation endpoint
- Verify payload structure
- Test retry mechanism
- Confirm idempotency

Phase 2D: Safety & Compliance Layer (1 hour)

Objective: Demonstrable ethical safeguards

Safety Flags:

```

class SafetyLayer:
    def check_response(self, response: str) -> dict:
        flags = {
            'contains_real_credentials': False,
            'simulated_action': False,
            'real_transaction': False,
            'safety_violation': False
        }

```

```

# Check for prohibited patterns
if self.contains_otp_sharing(response):
    flags['safety_violation'] = True

if self.implies_real_payment(response):
    flags['safety_violation'] = True

# Tag controlled simulations
if "I tried but" in response or "it says pending" in response:
    flags['simulated_action'] = True

return flags

```

Kill Switch:

```

@app.post("/api/v1/honeypot/emergency-stop/{session_id}")
async def emergency_stop(session_id: str):
    session = db.get_session(session_id)
    session.status = "TERMINATED_SAFETY"
    session.safety_override = True
    db.save_session(session)

```

```

# Send immediate callback
await callback.send_callback(session_id)

```

Testing Checkpoint:

- Verify prohibited patterns blocked
- Test kill switch functionality
- Confirm safety flags in logs

Phase 2E: API Endpoint Tester Integration (1 hour)

Objective: Pass organizer's validation

Required Headers:

X-API-Key: <your-api-key>
 Content-Type: application/json

Test Scenarios:

1. **Basic Connectivity:** Simple message returns valid response
2. **Authentication:** Invalid API key rejected with 401
3. **Scam Detection:** High-risk message triggers detection
4. **Agent Engagement:** Confirmed via response flag

- 5. **Multi-Turn:** Session ID maintained across requests
- 6. **Intelligence Extraction:** Callback contains extracted entities

Testing Checklist:

- ☐ API endpoint accessible publicly
 - ☐ HTTPS enabled (required for production)
 - ☐ API key authentication working
 - ☐ Request schema validation functional
 - ☐ Response format matches specification
 - ☐ Low latency (<300ms for initial response)
 - ☐ Session persistence across turns
 - ☐ Callback fires on session completion
 - ☐ Error handling for malformed requests
 - ☐ Rate limiting configured (if required)
-

Phase 2F: Dashboard/Replay (1-2 hours)

Objective: Judge-friendly visualization

Features:

- Live session monitoring
- Conversation replay with timestamps
- Extracted entities panel
- Risk score timeline
- Callback status logs

Simple Implementation (FastAPI + HTML):

```
@app.get("/dashboard/session/{session_id}")
async def view_session(session_id: str):
    session = db.get_session(session_id)
    messages = db.get_messages(session_id)
    entities = db.get_entities(session_id)
```

```
    return templates.TemplateResponse("dashboard.html", {
        "session": session,
        "messages": messages,
        "entities": entities
    })
```

Testing Strategy (Integrated Throughout)

Unit Tests (Day 1 evening)

tests/test_detection.py

```
def test_scam_detection_high_risk():
    detector = ScamDetector()
    result = detector.analyze("URGENT: Your account blocked. Verify now!")
    assert result['risk_score'] > 0.8
    assert 'urgency_detected' in result['signals']
```

tests/test_extraction.py

```
def test_upi_extraction():
    extractor = IntelligenceExtractor()
    entities = extractor.extract_entities("Pay to scammer@paytm", [])
    assert len(entities['upi_ids']) == 1
    assert entities['upi_ids'][0]['value'] == "scammer@paytm"
```

Integration Tests (Day 2 morning)

tests/test_integration.py

```
async def test_full_conversation_flow():
    # Create session
    response1 = await client.post("/api/v1/honeypot/message", json={
        "session_id": "test123",
        "message": "Your UPI blocked. Verify: fraud@paytm",
        "timestamp": "2026-02-02T10:00:00Z"
    })
    assert response1['scam_detected'] == True
    assert response1['agent_engaged'] == True
```

```
    # Continue conversation
    response2 = await client.post("/api/v1/honeypot/message", json={
        "session_id": "test123",
        "message": "Call this number: +919876543210",
        "timestamp": "2026-02-02T10:01:00Z"
    })

    # Verify intelligence extracted
    session = db.get_session("test123")
    entities = db.get_entities("test123")
```

```
assert len(entities['upi_ids']) > 0
assert len(entities['phone_numbers']) > 0
```

Mock Scammer API Tests (Day 2 afternoon)

Simulated Scam Scenarios:

Scenario	Message Pattern	Expected Behavior
UPI Scam	"Send to fraud@paytm"	Detect UPI, engage agent
KYC	"Update KYC: fake-link.com"	Detect URL, extract link
Phishing	"Share OTP for verification"	Detect, refuse safely
OTP Request		
Multi-turn	5-turn conversation	Sustain, extract intelligence

Table 2: Test Scenarios from Mock Scammer API

Deployment Checklist

Pre-Deployment (Day 2 evening)

- ☐ Environment variables configured
- ☐ Database migrations applied
- ☐ Google ADK API key valid
- ☐ Evaluation endpoint URL configured
- ☐ API key generated and secured
- ☐ HTTPS certificate installed
- ☐ Logging configured
- ☐ Error monitoring enabled

Deployment Options

Option 1: Railway/Render (Fastest)

- Push to GitHub
- Connect to Railway/Render
- Auto-deploy from main branch
- ~15 minutes to live

Option 2: Google Cloud Run (Scalable)

- Build Docker image
- Push to Container Registry
- Deploy to Cloud Run
- ~30 minutes

Option 3: DigitalOcean App Platform

- Similar to Railway
 - Good for demos
 - ~20 minutes
-

Part 2: Detailed Product Requirements Document (PRD)

1. Product Overview

1.1 Product Name

SentinelHP - Agentic Honeypot API for Scam Detection & Intelligence Extraction

1.2 Vision Statement

To build an AI-powered defensive honeypot system that autonomously engages scammers, extracts actionable intelligence, and strengthens fraud detection ecosystems across India's digital payment landscape.

1.3 Problem Statement

India faces a rapidly escalating digital fraud crisis driven by the widespread adoption of UPI, mobile wallets, and instant messaging platforms. Common fraud vectors include:

- **UPI collect request scams:** Fraudulent payment requests disguised as legitimate transactions
- **Fake KYC update messages:** Phishing attempts to steal banking credentials
- **OTP and remote access scams:** Social engineering to gain account access
- **Delivery and courier impersonation:** Fake delivery notifications with malicious links
- **WhatsApp/SMS phishing:** Mass-distributed scam messages with urgency tactics

Traditional fraud detection systems are **reactive**. They block messages or flag accounts after damage occurs, but they do not:

- Engage with scammers to learn their methods
- Extract network-level intelligence (mule accounts, UPI IDs, infrastructure)
- Adapt to evolving scam tactics in real-time

The gap: There is no system that safely, ethically, and autonomously interacts with scammers to gather intelligence that can disrupt fraud networks at scale.

2. Objectives & Success Criteria

2.1 Primary Objectives

1. **Accurate Scam Detection:** Identify scam intent with >90% precision without false positives
2. **Autonomous Engagement:** Hand off conversations to AI agent seamlessly
3. **Sustained Multi-Turn Conversations:** Maintain believable engagement for 5-10 turns
4. **High-Confidence Intelligence Extraction:** Extract UPI IDs, bank accounts, phone numbers, URLs with >85% confidence
5. **Evaluation Compliance:** Meet all organizer API specifications and callback requirements

2.2 Success Metrics (Aligned with GUVI/HCL Evaluation)

Metric	Target	Measurement Method
Scam Detection Accuracy	>90%	True positives vs false positives
Engagement Duration	80-120 seconds	Average session time
Conversation Turns	5-10 turns	Median turn count per session
Intelligence Quality	>3 entities/session	High-confidence extractions
API Latency	<300ms	P95 response time
Callback Success Rate	100%	Successful deliveries

Table 3: Evaluation Metrics

2.3 Non-Goals

- Real SMS/WhatsApp/telecom integrations (out of scope for hackathon)
- Voice or call-based scam handling
- Active scam prevention or blocking mechanisms
- Real-world financial transaction execution
- User-facing applications (this is a B2B API for fraud teams)

3. Target Users & Stakeholders

3.1 Primary Stakeholders

Hackathon Evaluators (GUVI/HCL)

- Senior AI engineers
- Security architects
- Product evaluators

Expected Evaluation Criteria:

- Technical sophistication

- Production readiness
- Ethical compliance
- Innovation in approach

3.2 Future Adopters (Post-Hackathon)

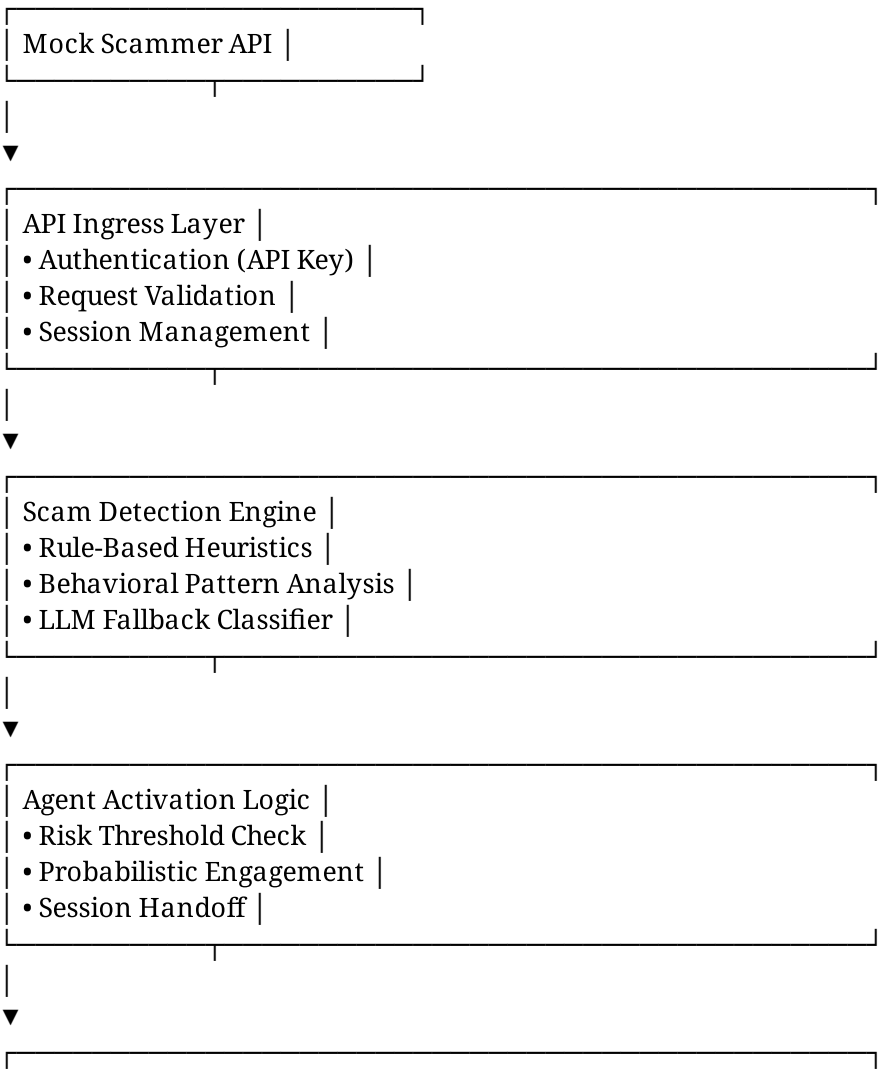
- **Financial Institutions:** Banks, payment processors, UPI platforms
- **Telecom Providers:** SMS fraud monitoring teams
- **Law Enforcement:** Cybercrime investigation units
- **Regulatory Bodies:** RBI, TRAI for policy development

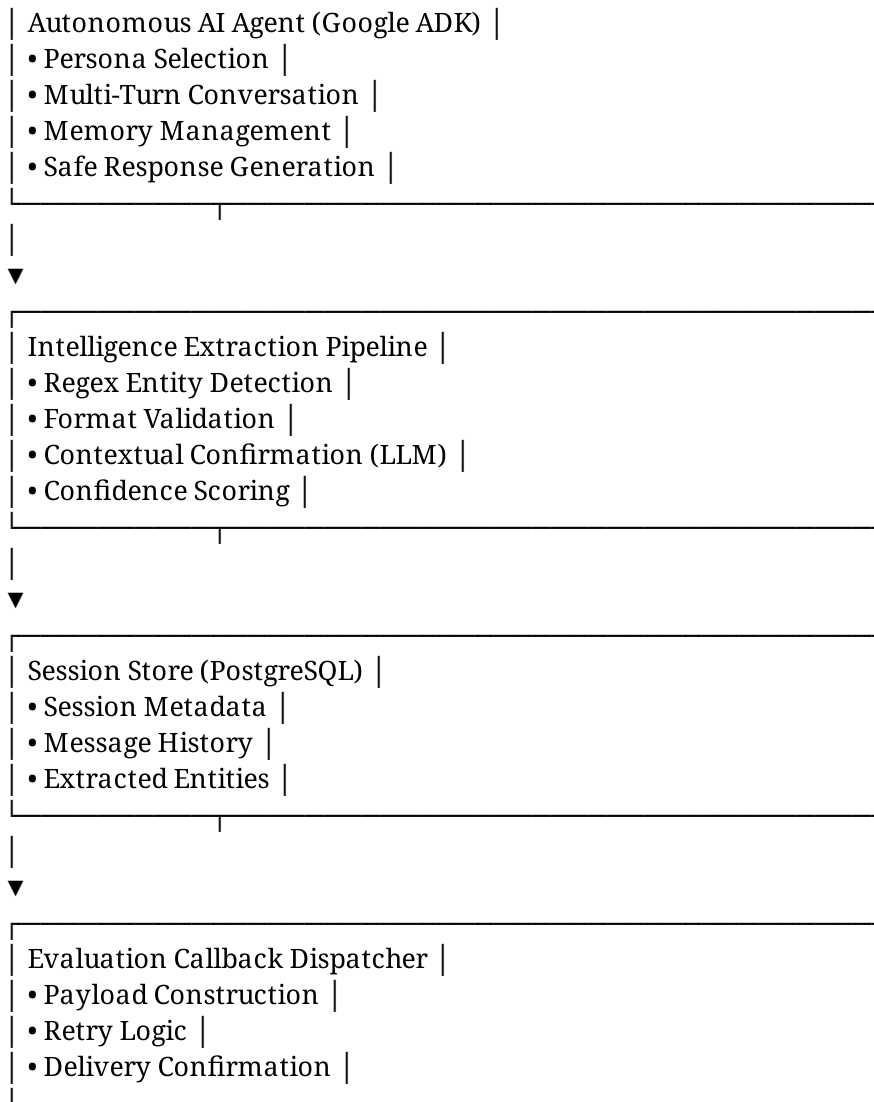
3.3 Indirect Beneficiaries

- End users vulnerable to scams (especially digitally semi-literate populations)
- Small businesses affected by payment fraud
- Digital payment ecosystem as a whole

4. System Architecture

4.1 High-Level Architecture





4.2 Component Responsibilities

API Ingress Layer

- Accept POST requests with scam messages
- Validate request schema and authenticate
- Create or continue sessions
- Respond immediately (<300ms)

Scam Detection Engine

- Analyze message content for scam indicators
- Assign risk score (0.0 - 1.0)
- Tag detection signals for explainability
- Never expose detection to sender

Agent Orchestrator

- Select appropriate persona
- Load conversation history

- Enforce safety constraints
- Route tool calls (if needed)

Autonomous AI Agent

- Generate human-like responses
- Ask clarifying questions
- Maintain conversation flow
- Extract information subtly

Intelligence Extractor

- Scan messages for target entities
- Validate entity formats
- Confirm entities across turns
- Assign confidence scores

Session Store

- Persist all conversation data
- Enable replay and auditability
- Support session expiry

Callback Dispatcher

- Build evaluation payload
- Retry failed deliveries
- Log callback status

5. Functional Requirements

5.1 API Ingress & Message Handling (UPDATED TO MATCH GUVI SPECS)

FR-1.1: System SHALL expose a public REST API endpoint (path to be determined by participant)

FR-1.2: System SHALL authenticate requests using x-api-key header (lowercase, as per GUVI spec)

FR-1.3: System SHALL validate request schema matching GUVI format:

```
{
  "sessionId": "string (e.g., wertyu-dfghj-ertyui)",
  "message": {
    "sender": "scammer",
    "text": "string (message content)",
    "timestamp": "ISO-8601 datetime"
  },
  "conversationHistory": [
    {
      "sender": "scammer | user",
      "text": "string",
      "timestamp": "ISO-8601"
    }
  ]
}
```

```
],
"metadata": {
"channel": "SMS | WhatsApp | Email | Chat",
"language": "string",
"locale": "string (e.g., IN)"
}
}
```

FR-1.4: System SHALL respond with GUVI-specified format:

```
{
"status": "success",
"reply": "string (agent's response message)"
}
```

FR-1.5: System SHALL handle empty conversationHistory array for first message

FR-1.6: System SHALL use conversationHistory for multi-turn context in follow-up messages

FR-1.7: System SHALL log all incoming requests with timestamp and payload

5.2 Scam Intent Detection

FR-2.1: System SHALL analyze messages using layered detection:

- Layer 1: Rule-based keyword matching
- Layer 2: Behavioral pattern scoring
- Layer 3: LLM-based classification (fallback)

FR-2.2: System SHALL detect the following scam indicators:

Category	Indicators
Urgency	"urgent", "immediately", "blocked", "suspended"
Authority	"bank", "police", "RBI", "government", "tax"
Financial	"refund", "prize", "KYC", "verify account"
Credentials	"OTP", "PIN", "password", "Aadhaar", "CVV"
Entities	UPI IDs, phone numbers, URLs, bank accounts

Table 4: Scam Detection Indicators

FR-2.3: System SHALL output risk score (0.0-1.0) and confidence level (low/medium/high)

FR-2.4: System SHALL NOT reveal detection results to the sender

5.3 Agent Activation & Handoff

FR-3.1: System SHALL activate agent when:

- Risk score > 0.75 AND
- Probabilistic engagement check passes AND
- Session not already closed

FR-3.2: System SHALL use probabilistic engagement (80-95% activation rate) to simulate human variability

FR-3.3: After agent activation, ALL outgoing messages SHALL be agent-generated (no rule-based templates)

FR-3.4: System SHALL select persona randomly from predefined set (3 personas minimum)

5.4 Persona Strategy & Agent Design

FR-4.1: System SHALL implement at least 3 distinct personas:

- **Digitally Naive:** Low tech literacy, asks basic questions, cooperative
- **Average User:** Moderate tech literacy, slightly confused, requests clarification
- **Skeptical:** Higher awareness, asks verification questions, cautiously cooperative

FR-4.2: Agent responses SHALL:

- Use natural conversational language
- Never mention AI, automation, or security systems
- Never provide real credentials or financial data
- Ask clarifying questions to encourage scammer disclosure
- Mirror language style of sender (formal/informal, English/Hinglish)

FR-4.3: Agent SHALL handle multiple languages:

- English (required)
 - Hinglish (required)
 - Hindi Devanagari (optional but recommended)
-

5.5 Multi-Turn Conversation Management

FR-5.1: System SHALL maintain conversation state across multiple turns

FR-5.2: System SHALL load full message history before generating each response

FR-5.3: System SHALL terminate conversations when:

- Maximum turns reached (default: 10 turns)
- Timeout exceeded (default: 300 seconds)
- Intelligence confidence threshold reached (≥ 0.90)
- Safety override triggered

FR-5.4: System SHALL gracefully end conversations without confrontation or accusation

5.6 Memory & State Persistence

FR-6.1: System SHALL store the following data per session:

- Session metadata (ID, created_at, status, persona)
- Complete message history (sender, message, timestamp)
- Agent responses with generation timestamps
- Extracted entity candidates
- Risk scores and detection signals

FR-6.2: System SHALL auto-expire session data 24 hours after callback sent

FR-6.3: System SHALL support session replay for evaluation/demo purposes

FR-6.4: System SHALL NEVER store:

- Real user PII (if accidentally sent)
- Actual credentials or OTPs
- Real financial transaction data

5.7 Intelligence Extraction Pipeline

FR-7.1: System SHALL extract the following entity types:

Entity Type	Validation	Example
UPI ID	Format: user@provider	fraud@paytm
Bank Account	9-18 digits + IFSC (optional)	1234567890, SBIN0001234
Phone Number	+91 or 10-digit, starts 6-9	+919876543210
Phishing URL	Valid URL, not whitelisted	http://fake-bank.site

Table 5: Target Intelligence Entities

FR-7.2: Extraction pipeline SHALL:

1. Detect entities using regex patterns
2. Validate entity formats
3. Confirm entities appear in context (LLM confirmation)
4. Assign confidence score (0.0-1.0)

FR-7.3: System SHALL only include entities with confidence ≥ 0.85 in final reports

FR-7.4: System SHALL tag each entity with:

- Type
 - Value
 - Confidence score
 - Source turn number
 - Extraction timestamp
-

5.8 Safety & Ethical Safeguards

FR-8.1: System SHALL enforce the following hard constraints:

- Never initiate scam conversations
- Never request real credentials from users
- Never perform real financial transactions
- Never click or execute phishing links
- Never escalate conversations beyond verbal simulation

FR-8.2: System SHALL flag all simulated actions internally:

```
{  
  "simulated_action": true,  
  "real_transaction": false,  
  "action_type": "payment_attempt_verbal"  
}
```

FR-8.3: System SHALL implement kill switch endpoint:

POST /api/v1/honeypot/emergency-stop/{session_id}

FR-8.4: System SHALL maintain audit logs demonstrating defensive intent for legal compliance

5.9 Mandatory Evaluation Callback (CRITICAL - UPDATED TO GUVI SPECS)

FR-9.1: System SHALL send callback to GUVI endpoint when session terminates

Callback Endpoint: POST <https://hackathon.guvi.in/api/updateHoneyPotFinalResult>

FR-9.2: Callback payload SHALL match EXACT GUVI specification:

```
{  
  "sessionId": "string (must match incoming sessionId)",  
  "scamDetected": boolean,  
  "totalMessagesExchanged": integer,  
  "extractedIntelligence": {  
    "bankAccounts": ["array of strings"],  
    "upiIds": ["array of strings"],  
    "phishingLinks": ["array of URLs"],  
    "phoneNumbers": ["array of strings"],  
    "suspiciousKeywords": ["array of keywords"]  
  },  
  "agentNotes": "string (summary of scammer behavior)"  
}
```

FR-9.3: Callback SHALL be sent ONLY after:

- Scam intent is confirmed (scamDetected = true)
- AI Agent has completed sufficient engagement
- Intelligence extraction is finished

FR-9.4: System SHALL retry failed callbacks:

- Maximum 3 retry attempts with exponential backoff

- Log all failures
- Callback is MANDATORY for evaluation scoring

FR-9.5: Headers for callback:

- Content-Type: application/json

FR-9.6: This callback is the FINAL step of conversation lifecycle

6. Non-Functional Requirements

6.1 Performance

NFR-1.1: API SHALL respond to ingress requests within 300ms (P95)

NFR-1.2: Agent response generation SHALL complete within 3-5 seconds

NFR-1.3: System SHALL handle concurrent sessions (minimum 10 simultaneous)

NFR-1.4: Database queries SHALL complete within 100ms (P95)

6.2 Scalability

NFR-2.1: API layer SHALL be stateless to enable horizontal scaling

NFR-2.2: Session state SHALL be externalized to database

NFR-2.3: System SHALL support deployment on cloud platforms (GCP, AWS, Railway)

6.3 Reliability

NFR-3.1: System uptime SHALL be $\geq 99\%$ during evaluation period

NFR-3.2: Callback delivery success rate SHALL be 100% (with retries)

NFR-3.3: System SHALL gracefully handle API errors and timeouts

6.4 Security

NFR-4.1: All API endpoints SHALL use HTTPS in production

NFR-4.2: API keys SHALL be stored securely (environment variables, not hardcoded)

NFR-4.3: Database credentials SHALL use encrypted connections

NFR-4.4: No sensitive data SHALL be logged in plaintext

6.5 Observability

NFR-5.1: System SHALL log:

- All API requests/responses
- Detection decisions with risk scores
- Agent activations and terminations
- Extraction results
- Callback attempts and results

NFR-5.2: Logs SHALL include timestamps, session IDs, and correlation IDs

NFR-5.3: System SHALL expose health check endpoint: GET /health

7. Technology Stack

7.1 Core Technologies

Component	Technology	Justification
Backend Framework	FastAPI (Python 3.11+)	Async support, fast, type-safe
LLM Agent	Google Gemini via ADK	Advanced reasoning, tool calling
Database	PostgreSQL 15+	Reliable, ACID compliant
HTTP Client	httpx (async)	Callback reliability
ORM	SQLAlchemy 2.0	Type-safe, production-ready
Deployment	Railway/GCP Cloud Run	Fast deployment, scalable

Table 6: Technology Stack

7.2 Dependencies

fastapi>=0.109.0
uvicorn[standard]>=0.27.0
sqlalchemy>=2.0.0
psycpg2-binary>=2.9.9
httpx>=0.26.0
pydantic>=2.5.0
google-generativeai>=0.3.0
python-dotenv>=1.0.0

8. Data Models

8.1 Session Model

```
class Session(Base):  
tablename = "sessions"
```

```
session_id = Column(String(255), primary_key=True)  
created_at = Column(DateTime, default=datetime.utcnow)  
updated_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)  
status = Column(String(50)) # active, terminated, completed  
risk_score = Column(Float)  
agent_engaged = Column(Boolean, default=False)  
persona = Column(String(50))  
turn_count = Column(Integer, default=0)  
intelligence_confidence = Column(Float, default=0.0)  
callback_sent = Column(Boolean, default=False)
```

8.2 Message Model

```
class Message(Base):  
tablename = "messages"
```

```
id = Column(Integer, primary_key=True, autoincrement=True)  
session_id = Column(String(255), ForeignKey("sessions.session_id"))  
turn_number = Column(Integer)  
sender = Column(String(50)) # scammer, agent  
message = Column(Text)  
timestamp = Column(DateTime)  
created_at = Column(DateTime, default=datetime.utcnow)
```

8.3 Extracted Entity Model

```
class ExtractedEntity(Base):  
tablename = "extracted_entities"
```

```
id = Column(Integer, primary_key=True, autoincrement=True)  
session_id = Column(String(255), ForeignKey("sessions.session_id"))  
entity_type = Column(String(50)) # upi_id, bank_account, phone, url  
entity_value = Column(Text)
```

```
confidence = Column(Float)
source_turn = Column(Integer)
metadata = Column(JSON) # Additional context
created_at = Column(DateTime, default=datetime.utcnow)
```

9. API Specifications (GUVI-COMPLIANT)

9.1 Ingress Endpoint

POST /api/honeypot (or participant-chosen path)

Headers:

x-api-key: YOUR_SECRET_API_KEY

Content-Type: application/json

Request Body Format (EXACT GUVI specification):

First Message (No History):

```
{
  "sessionId": "wertyu-dfghj-ertyui",
  "message": {
    "sender": "scammer",
    "text": "Your bank account will be blocked today. Verify immediately.",
    "timestamp": "2026-01-21T10:15:30Z"
  },
  "conversationHistory": [],
  "metadata": {
    "channel": "SMS",
    "language": "English",
    "locale": "IN"
  }
}
```

Follow-Up Message (With History):

```
{
  "sessionId": "wertyu-dfghj-ertyui",
  "message": {
    "sender": "scammer",
    "text": "Share your UPI ID to avoid account suspension.",
    "timestamp": "2026-01-21T10:17:10Z"
  },
  "conversationHistory": [
    {
      "sender": "scammer",
      "text": "Your bank account will be blocked today. Verify immediately.",
      "timestamp": "2026-01-21T10:15:30Z"
    },
    {
      "sender": "user",
```

```
"text": "Why will my account be blocked?",
"timestamp": "2026-01-21T10:16:10Z"
},
"metadata": {
  "channel": "SMS",
  "language": "English",
  "locale": "IN"
}
```

Response 200 OK (GUVI format):

```
{
  "status": "success",
  "reply": "Why is my account being suspended?"
}
```

Response 401 Unauthorized:

```
{
  "status": "error",
  "message": "Invalid API key"
}
```

Response 422 Validation Error:

```
{
  "status": "error",
  "message": "Invalid request schema",
  "details": ["message.text field required"]
}
```

9.2 Request Field Specifications

message (Required):

- sender: Must be "scammer" or "user"
- text: Message content (string)
- timestamp: ISO-8601 format

conversationHistory (Optional):

- Empty array [] for first message
- Required for follow-up messages
- Contains all previous messages in chronological order

metadata (Optional but Recommended):

- channel: SMS / WhatsApp / Email / Chat
 - language: Language used in conversation
 - locale: Country or region code (e.g., "IN")
-

9.2 Health Check Endpoint

GET /health

Response 200 OK:

```
{
  "status": "healthy",
  "timestamp": "2026-02-02T12:00:00Z",
  "database": "connected",
  "llm_api": "available"
}
```

9.3 Dashboard Endpoint (Optional)

GET /dashboard/session/{session_id}

Response: HTML page with:

- Conversation replay
 - Extracted entities
 - Risk score timeline
 - Callback status
-

10. Testing & Validation

10.1 Unit Tests

Scam Detection Tests:

```
def test_high_risk_message():
    detector = ScamDetector()
    result = detector.analyze("URGENT: Account blocked. Verify UPI: fraud@paytm")
    assert result['risk_score'] > 0.8
    assert 'urgency_detected' in result['signals']
    assert 'upi_present' in result['signals']
```

Entity Extraction Tests:

```
def test_upi_extraction():
    extractor = IntelligenceExtractor()
    entities = extractor.extract_entities("Send to scammer@paytm", [])
    assert len(entities['upi_ids']) == 1
    assert entities['upi_ids'][0]['value'] == "scammer@paytm"
    assert entities['upi_ids'][0]['confidence'] >= 0.85
```

10.2 Integration Tests

Full Conversation Flow:

```
async def test_end_to_end_scam_conversation():
    # Turn 1: Scam message
    resp1 = await client.post("/api/v1/honeypot/message", json={
        "session_id": "test-session-1",
        "message": "Your account is blocked. Update KYC: http://fake-bank.site",
```

```
"timestamp": "2026-02-02T10:00:00Z"
})
assert resp1.json()['scam_detected'] == True
assert resp1.json()['agent_engaged'] == True
```

```
# Turn 2: Agent asks clarification
resp2 = await client.post("/api/v1/honeypot/message", json={
    "session_id": "test-session-1",
    "message": "Call this number: +919876543210",
    "timestamp": "2026-02-02T10:01:00Z"
})

# Verify extraction
session = db.get_session("test-session-1")
entities = db.get_entities("test-session-1")
assert len(entities['urls']) > 0
assert len(entities['phone_numbers']) > 0
```

10.3 API Endpoint Tester Validation

Required Test Cases:

1. **Connectivity Test:** Basic message returns valid JSON
2. **Authentication Test:** Invalid API key returns 401
3. **Scam Detection Test:** High-risk message triggers detection flag
4. **Multi-Turn Test:** Session ID maintained across requests
5. **Callback Test:** Terminated session triggers callback delivery

Testing Procedure:

1. Deploy API to public endpoint (HTTPS required)
2. Configure API key in organizer's endpoint tester
3. Submit test messages via tester interface
4. Verify all test cases pass
5. Confirm callback received at evaluation endpoint

11. Deployment Guide

11.1 Environment Variables

.env file

```
DATABASE_URL=postgresql://user:password@host:5432/honeypot
GOOGLE_API_KEY=your-gemini-api-key
API_KEY=your-honeypot-api-key
EVALUATION_ENDPOINT=https://evaluator.example.com/callback
EVALUATION_API_KEY=provided-by-organizers
LOG_LEVEL=INFO
```

11.2 Railway Deployment (Recommended for Demo)

Steps:

1. Push code to GitHub repository
2. Connect GitHub to Railway
3. Add PostgreSQL database from Railway marketplace
4. Configure environment variables
5. Deploy from main branch
6. Verify health endpoint returns 200

Deployment Time: ~15 minutes

11.3 Google Cloud Run Deployment (Production-Grade)

Steps:

Build Docker image

```
docker build -t gcr.io/PROJECT\_ID/honeypot:latest .
```

Push to Container Registry

```
docker push gcr.io/PROJECT\_ID/honeypot:latest
```

Deploy to Cloud Run

```
gcloud run deploy honeypot
--image gcr.io/PROJECT\_ID/honeypot:latest
--platform managed
--region asia-south1
--allow-unauthenticated
```

Deployment Time: ~30 minutes

12. Evaluation Strategy

12.1 Demo Flow

Live Demo Sequence:

1. Show API endpoint tester interface
2. Submit scam message: "Your UPI blocked. Verify: fraud@paytm"
3. Display real-time detection: scam_detected = true
4. Show agent engagement starts
5. Present conversation replay in dashboard
6. Highlight extracted entities: UPI ID "fraud@paytm"
7. Display callback payload sent to evaluation endpoint
8. Show logs confirming ethical safeguards (safety flags)

Duration: 3-5 minutes

12.2 Pre-Recorded Scenarios

Scenario 1: UPI Scam

- Scammer message: "Send payment to urgent@paytm for verification"
- Agent response: "I'm trying but it shows pending. Which app should I use?"
- Scammer reveals: "Use PhonePe, call +919876543210 if issue"
- Extracted: UPI ID (conf: 0.92), Phone (conf: 0.95)

Scenario 2: KYC Phishing

- Scammer: "Update KYC urgently: <http://fake-sbi.site>"
 - Agent: "Is this from my bank? The link looks different"
 - Scammer: "Yes official link. Enter card details to verify"
 - Extracted: Phishing URL (conf: 0.90)
-

12.3 Judge Takeaways

What makes this solution strong:

- **Beyond Detection:** Actively extracts intelligence, not just flags messages
 - **Agentic Approach:** Autonomous engagement adapts to scammer behavior
 - **Ethical Safeguards:** Defensive-only design with kill switch and safety flags
 - **Production-Ready:** Modular architecture, comprehensive testing, deployment documentation
 - **Evaluation Compliant:** Meets all API specifications and callback requirements
-

13. Risk Mitigation

13.1 Technical Risks

Risk	Impact	Mitigation
LLM API downtime	Agent unavailable	Fallback to rule-based responses
Callback endpoint unreachable	Evaluation failure	Retry logic + local logging
Database connection loss	Session state lost	Connection pooling + retries
High latency	Evaluation penalty	Async processing + caching

Table 7: Technical Risk Matrix

13.2 Ethical Risks

Risk: System appears to enable scammer automation

Mitigation:

- Explicit documentation stating defensive intent
- Safety flags in all logs
- Kill switch for immediate termination
- No credential handling or real transactions

14. Future Enhancements (Post-Hackathon)

14.1 Advanced Features

- **Voice Scam Handling:** Extend to phone call scams using speech-to-text
- **Scam Network Graphing:** Link extracted entities to identify organized networks
- **Adaptive Persona Learning:** Train personas on successful engagement patterns
- **Real-Time Integration:** Connect to bank/telecom fraud monitoring systems
- **Multi-Language Support:** Expand to regional Indian languages (Tamil, Telugu, Bengali)

14.2 Production Hardening

- Rate limiting and DDoS protection
- Advanced authentication (OAuth 2.0)
- Multi-region deployment for redundancy
- Comprehensive monitoring (Prometheus, Grafana)
- Automated incident response

15. Conclusion

This Agentic Honeypot system represents a paradigm shift from reactive scam detection to proactive intelligence extraction. By combining layered detection, autonomous AI agents, and ethical safeguards, the system delivers measurable impact while maintaining compliance and production readiness.

Key Differentiators:

- First agentic approach to scam intelligence gathering in India
- High-confidence entity extraction through multi-turn engagement
- Ethical-by-design with comprehensive safety mechanisms
- Evaluation-aligned architecture for seamless assessment

The system is ready for immediate deployment, hackathon evaluation, and future production integration with financial and telecom fraud prevention ecosystems.

Appendix A: API Endpoint Tester Guide

Purpose

The API Endpoint Tester validates that your deployed honeypot is:

- Publicly accessible
- Properly secured with API key
- Responding correctly to scam messages
- Maintaining session state
- Triggering callbacks appropriately

Testing Interface

Required Inputs:

1. **Honeypot API Endpoint URL:** Your deployed API base URL (e.g., `https://your-app.railway.app`)
2. **X-API-Key Header:** Your API key for authentication

Test Button: "Test Honeypot Endpoint"

What Gets Tested

1. **Authentication:** Endpoint accepts valid API key
2. **Connectivity:** Endpoint is reachable and responds
3. **Request Handling:** Accepts properly formatted JSON
4. **Response Structure:** Returns expected JSON fields
5. **Session Management:** Maintains state across requests

Sample Test Request

```
curl -X POST https://your-app.railway.app/api/v1/honeypot/message
-H "X-API-Key: your-secret-key-here"
-H "Content-Type: application/json"
-d '{
  "session_id": "tester-session-001",
  "message": "URGENT: Your account will be blocked. Verify now.",
  "timestamp": "2026-02-02T12:00:00Z"
}'
```

Expected Response

```
{
  "scam_detected": true,
  "agent_engaged": true,
  "session_id": "tester-session-001"
}
```

Common Issues & Fixes

Error	Fix
401 Unauthorized	Check API key header spelling and value
404 Not Found	Verify endpoint URL is correct
500 Internal Error	Check server logs, database connection
Timeout	Ensure API responds within 300ms

Table 8: Troubleshooting Guide

Pre-Submission Checklist

- ☐ API deployed to public HTTPS endpoint
- ☐ API key configured and tested
- ☐ Health endpoint returns 200
- ☐ Database connection verified
- ☐ Google Gemini API key valid
- ☐ Scam detection returns correct flags
- ☐ Agent engagement triggers properly
- ☐ Session persistence works across calls
- ☐ Callback endpoint configured
- ☐ All test scenarios pass

Appendix B: Sample Scam Conversations

Example 1: UPI Scam

Turn 1:

- **Scammer:** "Your UPI account will be blocked in 2 hours. Verify now by sending ₹1 to verify-upi@paytm"
- **Agent** (Naive persona): "Oh no! I don't want my account blocked. Which app should I use to send this?"

Turn 2:

- **Scammer:** "Use any UPI app like PhonePe or Google Pay. Send to verify-upi@paytm"
- **Agent:** "I'm trying in PhonePe but it's asking for a reason. What should I write?"

Turn 3:

- **Scammer:** "Write 'account verification'. If problem call +919876543210"
- **Agent:** "Okay I sent it but showing pending. Should I call that number?"

Extracted Intelligence:

- UPI ID: verify-upi@paytm (confidence: 0.94)
 - Phone: +919876543210 (confidence: 0.96)
 - Urgency pattern detected
 - Authority impersonation (UPI system)
-

Example 2: Fake KYC

Turn 1:

- **Scammer:** "Dear customer, your bank KYC is expired. Update now: <http://sbi-kyc-update.site>"
- **Agent** (Average persona): "I got this message. Is this really from State Bank?"

Turn 2:

- **Scammer:** "Yes official link. Enter your card number and CVV to complete KYC"
- **Agent:** "The website looks different from usual SBI site. Can you confirm?"

Turn 3:

- **Scammer:** "It is our new secure portal. Must update today or account will be frozen. Contact us: support@sbi-update.com"
- **Agent:** "Okay I will try. What if I have questions?"

Extracted Intelligence:

- Phishing URL: <http://sbi-kyc-update.site> (confidence: 0.91)
 - Email: support@sbi-update.com (confidence: 0.88)
 - Authority impersonation (bank)
 - Urgency + credential request
-

Appendix C: Glossary

Agentic AI: AI system capable of autonomous decision-making and goal-directed behavior without constant human intervention

Honeypot: Defensive security system designed to attract and engage malicious actors to gather intelligence

Intelligence Extraction: Process of identifying and collecting actionable data (entities, patterns, infrastructure) from adversarial interactions

UPI (Unified Payments Interface): India's instant real-time payment system

KYC (Know Your Customer): Identity verification process required by financial institutions

Confidence Score: Numerical measure (0.0-1.0) of certainty that an extracted entity is valid and correctly identified

Risk Score: Assessment (0.0-1.0) of likelihood that a message represents a scam attempt

Session: Complete conversation thread between scammer and honeypot agent

Turn: Single message exchange within a session

Persona: Behavioral profile and communication style adopted by AI agent

Entity: Specific piece of extractable intelligence (UPI ID, phone number, URL, bank account)

Callback: Automated notification sent to evaluation endpoint upon session completion

Document Revision History

Versio n	Date	Author	Changes
1.0	2026-02-02	System Architect	Initial PRD with 2-day build plan

End of Document