# APPENDIX

## AI-Health Platform - Technical Reference

*Supplementary documentation for the AI-Health Care Plan Platform*

---

## Table of Contents



---

## Appendix A: Sample Doctor Prescriptions

### Example 1: Simple Respiratory Infection

**Use Case:** Common outpatient scenario for a young adult with mild comorbidity


```
{
  "patient_info": {
    "age": 32,
    "gender": "Male",
    "weight": 75.0,
    "medical_conditions": ["Mild asthma"],
    "allergies": ["Shellfish"]
  },
  "diagnosis": "Upper respiratory tract infection",
  "prescriptions": [
    {
      "medication_name": "Azithromycin",
      "dosage": "500mg on day 1, then 250mg daily",
      "duration": "5 days",
      "instructions": "Take on empty stomach, 1 hour before or 2 hours
after meals"
    },
    {
      "medication_name": "Guaifenesin",
      "dosage": "400mg every 4 hours",
```

```
      "duration": "7 days",
      "instructions": "Take with plenty of water to help loosen mucus"
    },
    {
      "medication_name": "Acetaminophen",
      "dosage": "500mg as needed",
      "duration": "Until fever resolves",
      "instructions": "Do not exceed 3000mg in 24 hours. Take for fever
above 100.4°F"
    }
  ],
  "doctor_notes": "Patient presents with productive cough, fever
(101.2°F), and congestion for 3 days. Monitor asthma symptoms closely.
Continue regular asthma medications. Return if symptoms worsen or no
improvement in 3 days.",
  "prescription_date": "2025-11-08T09:30:00"
}
```

**Expected AI Processing:**

- **Model Recommendation**: Claude 3.5 Sonnet or Nova Micro
- **Processing Time**: 3-5 seconds
- **Risk Alerts**: Low (monitoring for shellfish allergy, asthma interaction)
- **Care Plan Sections**: 7 comprehensive sections

---

### Example 2: Complex Multi-Comorbidity Case

**Use Case:** Elderly patient with multiple chronic conditions requiring specialized care planning

```
{
  "patient_info": {
    "age": 68,
    "gender": "Female",
    "weight": 68.0,
    "medical_conditions": [
      "Hypertension (15 years)",
      "Type 2 Diabetes Mellitus (10 years)",
      "Chronic Kidney Disease Stage 3b",
      "Atrial Fibrillation",
      "Hyperlipidemia"
    ],
    "allergies": ["Penicillin (anaphylaxis)", "Iodine contrast dye",
"Sulfa drugs"]
  },
  "diagnosis": "Acute decompensated heart failure with reduced ejection
fraction (HFrEF) - NYHA Class III",
  "prescriptions": [
    {
      "medication_name": "Furosemide (Lasix)",
      "dosage": "40mg twice daily (morning and early afternoon)",
```

```
      "duration": "14 days initially, then reassess",
      "instructions": "Take in morning (8am) and early afternoon (2pm) to
avoid nighttime urination. Monitor daily weight. Call if weight increases
>2 lbs in 24 hours or >5 lbs in one week."
    },
    {
      "medication_name": "Metoprolol Succinate ER",
      "dosage": "25mg once daily",
      "duration": "Ongoing - long term",
      "instructions": "Check pulse before taking. Do not take if heart
rate <55 bpm. Take at same time daily, preferably morning."
    },
    {
      "medication_name": "Lisinopril",
      "dosage": "5mg once daily",
      "duration": "Ongoing - long term",
      "instructions": "Monitor blood pressure and kidney function. May
cause dizziness - rise slowly from sitting/lying. Avoid salt substitutes
containing potassium."
    },
    {
      "medication_name": "Metformin",
      "dosage": "500mg twice daily",
      "duration": "Ongoing",
      "instructions": "Take with meals to reduce GI upset. Monitor kidney
function regularly due to CKD."
    },
    {
      "medication_name": "Apixaban (Eliquis)",
      "dosage": "5mg twice daily",
      "duration": "Ongoing",
      "instructions": "Blood thinner for AFib. Take exactly 12 hours
apart. Report any unusual bleeding. No skipping doses."
    },
    {
      "medication_name": "Atorvastatin",
      "dosage": "40mg at bedtime",
      "duration": "Ongoing",
      "instructions": "Take at night for optimal effectiveness. Report
muscle pain or weakness."
    },
    {
      "medication_name": "Potassium Chloride",
      "dosage": "20 mEq once daily",
      "duration": "While on Furosemide",
      "instructions": "Supplement due to diuretic use. Take with food and
full glass of water. Will monitor blood potassium levels."
    }
  ],
  "doctor_notes": "Patient admitted to ER with dyspnea at rest, orthopnea
(3 pillow), peripheral edema (3+ pitting to mid-calf bilateral), JVD, and
weight gain of 8 lbs over 5 days. BNP 1250 pg/mL, Creatinine 1.9 mg/dL
(baseline 1.6), GFR 32, K+ 3.8. Echo shows EF 30% (previous 35%). CXR with
pulmonary congestion. Discharged after 3-day admission with IV diuresis.
```

```
Close monitoring essential due to narrow therapeutic window with CKD and
multiple comorbidities. High risk for drug interactions and electrolyte
imbalances. Follow-up in 3 days with labs. Home health nurse referral
placed. Patient educated on daily weights, fluid restriction (1.5L/day),
sodium restriction (<2g/day), and warning signs requiring immediate
attention.",
  "prescription_date": "2025-11-07T16:45:00"
}
```

**Expected AI Processing:**

- **Model Recommendation**: Claude 4.5 Sonnet or Amazon Nova Micro (medical factors specialist)
- **Processing Time**: 5-8 seconds
- **Risk Alerts**: High/Moderate (polypharmacy, drug interactions, CKD considerations, age-related risks)
- **Care Plan Complexity**: Advanced with detailed monitoring schedules

**Key Risk Factors Detected:**

1. ⚠ **Polypharmacy** - 7 medications simultaneously
2. ⚠ **Drug Interaction** - Lisinopril + Potassium (hyperkalemia risk)
3. ⚠ **CKD Risk** - Metformin with GFR 32 (dose adjustment needed)
4. ⚠ **Age-Related** - Elderly patient with multiple comorbidities
5. ⚠ **Narrow Therapeutic Window** - Complex medication balancing required

---

### Example 3: Pediatric Case

**Use Case:** Pediatric patient requiring weight-based dosing verification

```
{
  "patient_info": {
    "age": 7,
    "gender": "Female",
    "weight": 23.0,
    "medical_conditions": ["Seasonal allergies"],
    "allergies": ["None known"]
  },
  "diagnosis": "Acute otitis media (middle ear infection) - bilateral",
  "prescriptions": [
    {
      "medication_name": "Amoxicillin",
      "dosage": "400mg (8mL of 100mg/5mL suspension) twice daily",
      "duration": "10 days",
      "instructions": "Give with or without food. Shake suspension well
before each use. Complete full 10-day course even if symptoms improve.
Refrigerate suspension."
    },
    {
      "medication_name": "Ibuprofen",
      "dosage": "150mg (7.5mL of children's suspension) every 6 hours as
needed",
```

```
      "duration": "3-5 days or until pain resolves",
      "instructions": "For pain and fever. Give with food or milk. Do not
exceed 4 doses in 24 hours."
    }
  ],
  "doctor_notes": "Child presents with 2 days of ear pain, fever to
102.5°F, and decreased appetite. Examination shows bilateral TM bulging
with reduced mobility. No perforation noted. Parent reports recent upper
respiratory infection. Child attends daycare. Follow-up in 2 weeks or
sooner if symptoms worsen. Watch for signs of allergic reaction (rash,
hives, difficulty breathing). Educate parent on proper antibiotic
administration and importance of completing full course.",
  "prescription_date": "2025-11-08T14:15:00"
}
```

**Expected AI Processing:**

- **Model Recommendation**: Nova Micro or Claude 3.5 Sonnet
- **Processing Time**: 3-5 seconds
- **Risk Alerts**: Low/Moderate (pediatric dosing verification)
- **Special Considerations**: Parent education, adherence instructions

**Pediatric-Specific Elements:**

- ✅ Weight-based dosing calculation (Amoxicillin: ~17mg/kg/dose)
- ✅ Liquid formulation instructions
- ✅ Parent-friendly care instructions
- ✅ Signs to watch for allergic reactions
- ✅ Follow-up timing appropriate for age

---

# Appendix B: Python Code for Risk Detection

## Risk Detection and Validation Module

This module provides comprehensive prescription safety analysis before care plan generation.

```python
"""
Risk Detection Module for AI-Health Platform
Identifies high-risk scenarios and validates prescription safety
"""

import logging
from typing import Dict, List, Optional, Tuple
from dataclasses import dataclass
from enum import Enum

logger = logging.getLogger(__name__)


class RiskLevel(Enum):
```

```
    """Risk severity levels"""
    LOW = "low"
    MODERATE = "moderate"
    HIGH = "high"
    CRITICAL = "critical"


@dataclass
class RiskAlert:
    """Risk alert structure"""
    risk_level: RiskLevel
    category: str
    message: str
    recommendation: str
    requires_immediate_action: bool = False


class PrescriptionRiskDetector:
    """
    Detects potential risks in prescriptions based on:
    - Drug interactions
    - Patient age and comorbidities
    - Dosage appropriateness
    - Kidney/liver function considerations
    - Allergy conflicts
    """

    # High-risk drug combinations
    DRUG_INTERACTIONS = {
        ("warfarin", "aspirin"): {
            "risk": RiskLevel.HIGH,
            "message": "Increased bleeding risk with concurrent use",
            "recommendation": "Monitor INR closely, consider alternative
antiplatelet"
        },
        ("metformin", "furosemide"): {
            "risk": RiskLevel.MODERATE,
            "message": "Increased risk of lactic acidosis with diuretics",
            "recommendation": "Monitor kidney function and lactate levels"
        },
        ("lisinopril", "potassium"): {
            "risk": RiskLevel.HIGH,
            "message": "Risk of hyperkalemia with ACE inhibitors and K+
supplements",
            "recommendation": "Monitor potassium levels weekly initially"
        },
        ("digoxin", "furosemide"): {
            "risk": RiskLevel.HIGH,
            "message": "Diuretics can cause hypokalemia, increasing
digoxin toxicity risk",
            "recommendation": "Monitor potassium and digoxin levels
regularly"
        }
    }
```

```
    # Kidney function considerations
    CKD_CONTRAINDICATED = [
        "metformin",  # CI if GFR < 30
        "nsaids",
        "contrast_dye"
    ]

    # Age-related risk factors
    ELDERLY_HIGH_RISK = [
        "benzodiazepines",
        "anticholinergics",
        "nsaids",
        "high_dose_opioids"
    ]

    def __init__(self):
        """Initialize risk detector"""
        self.risk_alerts: List[RiskAlert] = []

    def analyze_prescription(self, prescription_data: Dict) ->
List[RiskAlert]:
        """
        Comprehensive risk analysis of prescription

        Args:
            prescription_data: Doctor prescription dictionary

        Returns:
            List of risk alerts
        """
        self.risk_alerts = []

        # Extract data
        patient_info = prescription_data.get("patient_info", {})
        prescriptions = prescription_data.get("prescriptions", [])
        medical_conditions = patient_info.get("medical_conditions", [])
        allergies = patient_info.get("allergies", [])
        age = patient_info.get("age", 0)

        # Run all risk checks
        self._check_drug_interactions(prescriptions)
        self._check_allergy_conflicts(prescriptions, allergies)
        self._check_age_appropriateness(age, prescriptions)
        self._check_comorbidity_risks(medical_conditions, prescriptions)
        self._check_kidney_function(medical_conditions, prescriptions)
        self._check_polypharmacy(prescriptions)
        self._check_dosage_safety(prescriptions, patient_info)

        return self.risk_alerts

    def _check_drug_interactions(self, prescriptions: List[Dict]) -> None:
        """Check for known drug-drug interactions"""
        medications = [p["medication_name"].lower() for p in
```

```
prescriptions]

        for i, med1 in enumerate(medications):
            for med2 in medications[i+1:]:
                # Check both orderings
                interaction = (
                    self.DRUG_INTERACTIONS.get((med1, med2)) or
                    self.DRUG_INTERACTIONS.get((med2, med1))
                )

                if interaction:
                    self.risk_alerts.append(RiskAlert(
                        risk_level=interaction["risk"],
                        category="Drug Interaction",
                        message=f"{med1.title()} + {med2.title()}:
{interaction['message']}",
                        recommendation=interaction["recommendation"],
                        requires_immediate_action=interaction["risk"] ==
RiskLevel.CRITICAL
                    ))

    def _check_allergy_conflicts(self, prescriptions: List[Dict],
                                 allergies: List[str]) -> None:
        """Check if any prescribed medication conflicts with known
allergies"""
        if not allergies:
            return

        allergy_keywords = [a.lower() for a in allergies]

        for prescription in prescriptions:
            med_name = prescription["medication_name"].lower()

            for allergy in allergy_keywords:
                if allergy in med_name or
self._check_drug_class_allergy(med_name, allergy):
                    self.risk_alerts.append(RiskAlert(
                        risk_level=RiskLevel.CRITICAL,
                        category="Allergy Conflict",
                        message=f"CONTRAINDICATED:
{prescription['medication_name']} - Patient has documented {allergy}
allergy",
                        recommendation="DO NOT PRESCRIBE. Select
alternative medication immediately.",
                        requires_immediate_action=True
                    ))

    def _check_drug_class_allergy(self, medication: str, allergy: str) ->
bool:
        """Check if medication belongs to allergy class"""
        drug_classes = {
            "penicillin": ["amoxicillin", "ampicillin", "penicillin"],
            "sulfa": ["sulfamethoxazole", "trimethoprim", "furosemide"],
            "nsaid": ["ibuprofen", "naproxen", "aspirin", "diclofenac"]
```

```
        }

        if allergy in drug_classes:
            return any(drug in medication for drug in
drug_classes[allergy])
        return False

    def _check_age_appropriateness(self, age: int, prescriptions:
List[Dict]) -> None:
        """Check if medications are appropriate for patient age"""

        # Elderly patients (≥65)
        if age >= 65:
            for prescription in prescriptions:
                med_name = prescription["medication_name"].lower()

                if any(risk_med in med_name for risk_med in
self.ELDERLY_HIGH_RISK):
                    self.risk_alerts.append(RiskAlert(
                        risk_level=RiskLevel.HIGH,
                        category="Age-Related Risk",
                        message=f"Beers Criteria:
{prescription['medication_name']} potentially inappropriate in elderly
(age {age})",
                        recommendation="Consider alternative with better
safety profile in elderly. If necessary, use lowest effective dose and
monitor closely."
                    ))

        # Pediatric patients (<18)
        if age < 18:
            self._check_pediatric_dosing(age, prescriptions)

    def _check_pediatric_dosing(self, age: int, prescriptions: List[Dict])
-> None:
        """Verify pediatric dosing is weight-based and appropriate"""
        for prescription in prescriptions:
            dosage = prescription.get("dosage", "").lower()

            # Check if dosage mentions mg/kg (weight-based)
            if age < 12 and "mg/kg" not in dosage and "ml" not in dosage:
                self.risk_alerts.append(RiskAlert(
                    risk_level=RiskLevel.MODERATE,
                    category="Pediatric Dosing",
                    message=f"Pediatric patient (age {age}): Verify
weight-based dosing for {prescription['medication_name']}",
                    recommendation="Confirm dosage is appropriate for
patient weight. Consider using mg/kg calculation."
                ))

    def _check_comorbidity_risks(self, conditions: List[str],
                                 prescriptions: List[Dict]) -> None:
        """Check for medication risks with existing conditions"""
        condition_lower = [c.lower() for c in conditions]
```

```
        # Heart failure + NSAIDs
        if any("heart failure" in c for c in condition_lower):
            for prescription in prescriptions:
                med = prescription["medication_name"].lower()
                if any(nsaid in med for nsaid in ["ibuprofen", "naproxen",
"nsaid"]):
                    self.risk_alerts.append(RiskAlert(
                        risk_level=RiskLevel.HIGH,
                        category="Comorbidity Risk",
                        message="NSAIDs can worsen heart failure - fluid
retention and increased BP",
                        recommendation="Use acetaminophen for pain/fever.
If NSAID necessary, use lowest dose for shortest duration with close
monitoring."
                    ))

        # Diabetes + beta blockers
        if any("diabetes" in c for c in condition_lower):
            for prescription in prescriptions:
                med = prescription["medication_name"].lower()
                if "metoprolol" in med or "propranolol" in med or
"atenolol" in med:
                    self.risk_alerts.append(RiskAlert(
                        risk_level=RiskLevel.MODERATE,
                        category="Comorbidity Risk",
                        message="Beta blockers can mask hypoglycemia
symptoms in diabetic patients",
                        recommendation="Educate patient on hypoglycemia
awareness. Monitor blood glucose more frequently."
                    ))

    def _check_kidney_function(self, conditions: List[str],
                               prescriptions: List[Dict]) -> None:
        """Check medication safety with kidney disease"""
        has_ckd = any("kidney" in c.lower() or "renal" in c.lower() for c
in conditions)

        if has_ckd:
            for prescription in prescriptions:
                med = prescription["medication_name"].lower()

                if any(contraindicated in med for contraindicated in
self.CKD_CONTRAINDICATED):
                    self.risk_alerts.append(RiskAlert(
                        risk_level=RiskLevel.HIGH,
                        category="Kidney Function Risk",
                        message=f"{prescription['medication_name']}
requires dose adjustment or contraindicated in CKD",
                        recommendation="Verify GFR and adjust dose
accordingly. Consider alternative if GFR <30. Monitor kidney function
closely."
                    ))
```

```
    def _check_polypharmacy(self, prescriptions: List[Dict]) -> None:
        """Check for polypharmacy risks (≥5 medications)"""
        if len(prescriptions) >= 5:
            self.risk_alerts.append(RiskAlert(
                risk_level=RiskLevel.MODERATE,
                category="Polypharmacy",
                message=f"Patient prescribed {len(prescriptions)}
medications - increased risk of adverse events",
                recommendation="Review each medication for continued
necessity. Assess for drug-drug interactions and cumulative side effects."
            ))

    def _check_dosage_safety(self, prescriptions: List[Dict],
                             patient_info: Dict) -> None:
        """Verify dosages are within safe ranges"""
        age = patient_info.get("age", 0)
        weight = patient_info.get("weight")

        for prescription in prescriptions:
            med = prescription["medication_name"].lower()
            dosage_str = prescription.get("dosage", "").lower()

            # Check for excessive acetaminophen
            if "acetaminophen" in med or "tylenol" in med:
                if "500mg" in dosage_str or "1000mg" in dosage_str:
                    self.risk_alerts.append(RiskAlert(
                        risk_level=RiskLevel.MODERATE,
                        category="Dosage Safety",
                        message="Acetaminophen: Ensure total daily dose
does not exceed 3000mg (4000mg max)",
                        recommendation="Advise patient to avoid other
acetaminophen-containing products (e.g., cold medications)"
                    ))

    def get_risk_summary(self) -> Dict[str, int]:
        """Get summary count of risks by level"""
        return {
            "critical": sum(1 for alert in self.risk_alerts if
alert.risk_level == RiskLevel.CRITICAL),
            "high": sum(1 for alert in self.risk_alerts if
alert.risk_level == RiskLevel.HIGH),
            "moderate": sum(1 for alert in self.risk_alerts if
alert.risk_level == RiskLevel.MODERATE),
            "low": sum(1 for alert in self.risk_alerts if alert.risk_level
== RiskLevel.LOW),
            "total": len(self.risk_alerts)
        }

    def has_critical_risks(self) -> bool:
        """Check if any critical risks present"""
        return any(alert.risk_level == RiskLevel.CRITICAL for alert in
self.risk_alerts)
```

```python
# Example usage
def analyze_prescription_safety(prescription_data: Dict) -> Tuple[bool,
List[RiskAlert], Dict]:
    """
    Analyze prescription for safety risks

    Returns:
        Tuple of (is_safe, risk_alerts, risk_summary)
    """
    detector = PrescriptionRiskDetector()
    alerts = detector.analyze_prescription(prescription_data)
    summary = detector.get_risk_summary()
    is_safe = not detector.has_critical_risks()

    return is_safe, alerts, summary


# Integration example with care plan generation
def generate_safe_care_plan(prescription_data: Dict) -> Dict:
    """
    Generate care plan with integrated risk detection
    """
    # First, check for safety risks
    is_safe, risk_alerts, risk_summary =
analyze_prescription_safety(prescription_data)

    if not is_safe:
        logger.warning(f"CRITICAL RISKS DETECTED:
{risk_summary['critical']} critical issues")
        return {
            "success": False,
            "error": "Critical safety risks detected - prescription
requires review",
            "risk_alerts": [
                {
                    "level": alert.risk_level.value,
                    "category": alert.category,
                    "message": alert.message,
                    "recommendation": alert.recommendation
                }
                for alert in risk_alerts
            ],
            "risk_summary": risk_summary
        }

    # If safe, proceed with AI care plan generation
    # ... (call to Bedrock API)

    return {
        "success": True,
        "care_plan": {},  # Generated care plan
        "risk_alerts": [
            {
                "level": alert.risk_level.value,
```

```
                "category": alert.category,
                "message": alert.message,
                "recommendation": alert.recommendation
            }
            for alert in risk_alerts if alert.risk_level in
[RiskLevel.HIGH, RiskLevel.MODERATE]
        ],
        "risk_summary": risk_summary
    }
```

### Usage Example

```
# Example 1: Analyze the complex multi-comorbidity case
from risk_detector import analyze_prescription_safety

prescription = {
    "patient_info": {
        "age": 68,
        "medical_conditions": ["Heart Failure", "CKD Stage 3b",
"Diabetes"],
        "allergies": ["Penicillin"]
    },
    "prescriptions": [
        {"medication_name": "Lisinopril", "dosage": "5mg daily"},
        {"medication_name": "Potassium Chloride", "dosage": "20 mEq
daily"},
        {"medication_name": "Metformin", "dosage": "500mg twice daily"},
        # ... more medications
    ]
}

is_safe, alerts, summary = analyze_prescription_safety(prescription)

print(f"Prescription Safe: {is_safe}")
print(f"Total Alerts: {summary['total']}")
print(f"Critical: {summary['critical']}, High: {summary['high']},
Moderate: {summary['moderate']}")

for alert in alerts:
    print(f"\n[{alert.risk_level.value.upper()}] {alert.category}")
    print(f"  {alert.message}")
    print(f"  Recommendation: {alert.recommendation}")
```

**Output:**

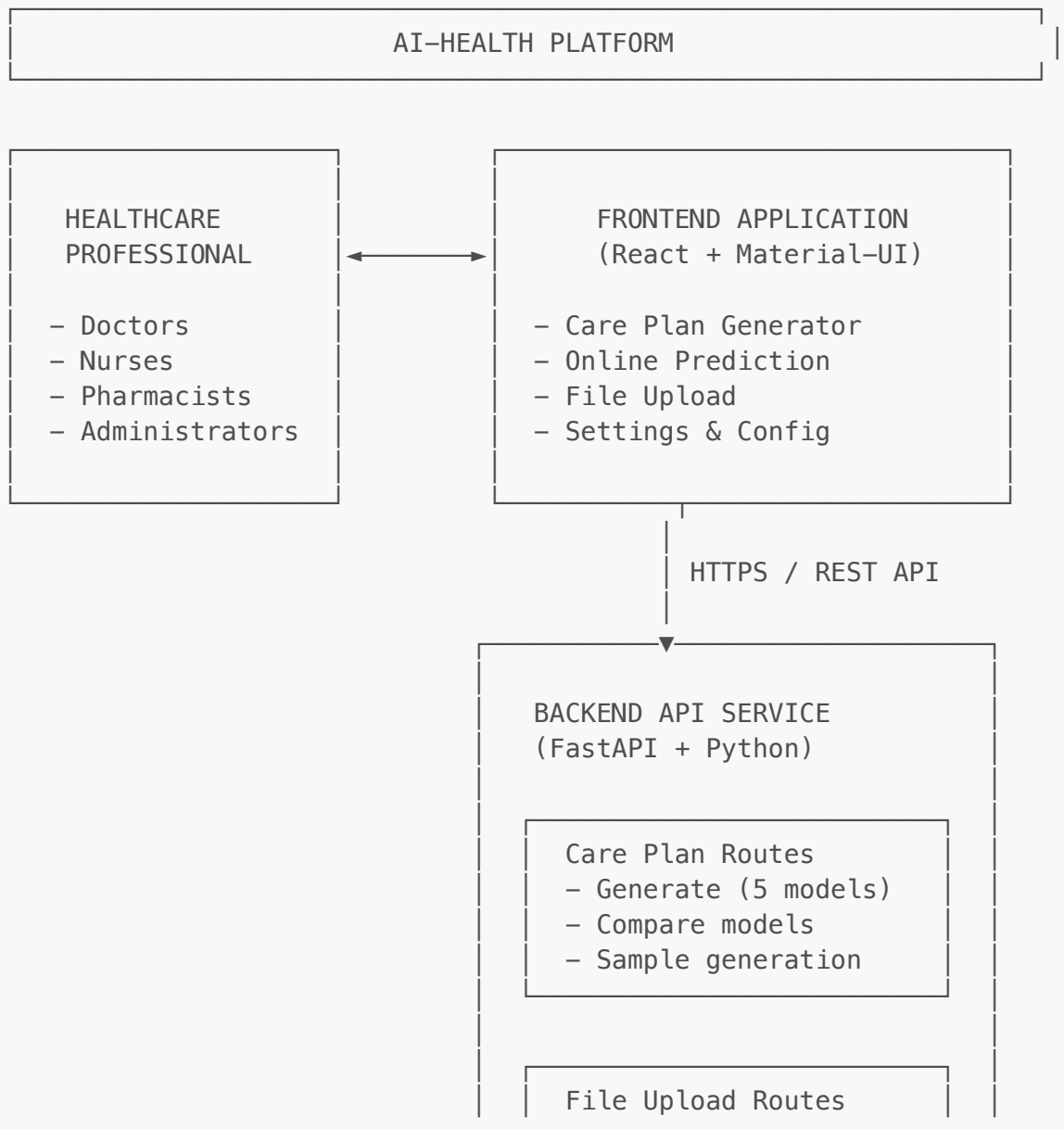
```
Prescription Safe: False
Total Alerts: 3
Critical: 0, High: 2, Moderate: 1
```

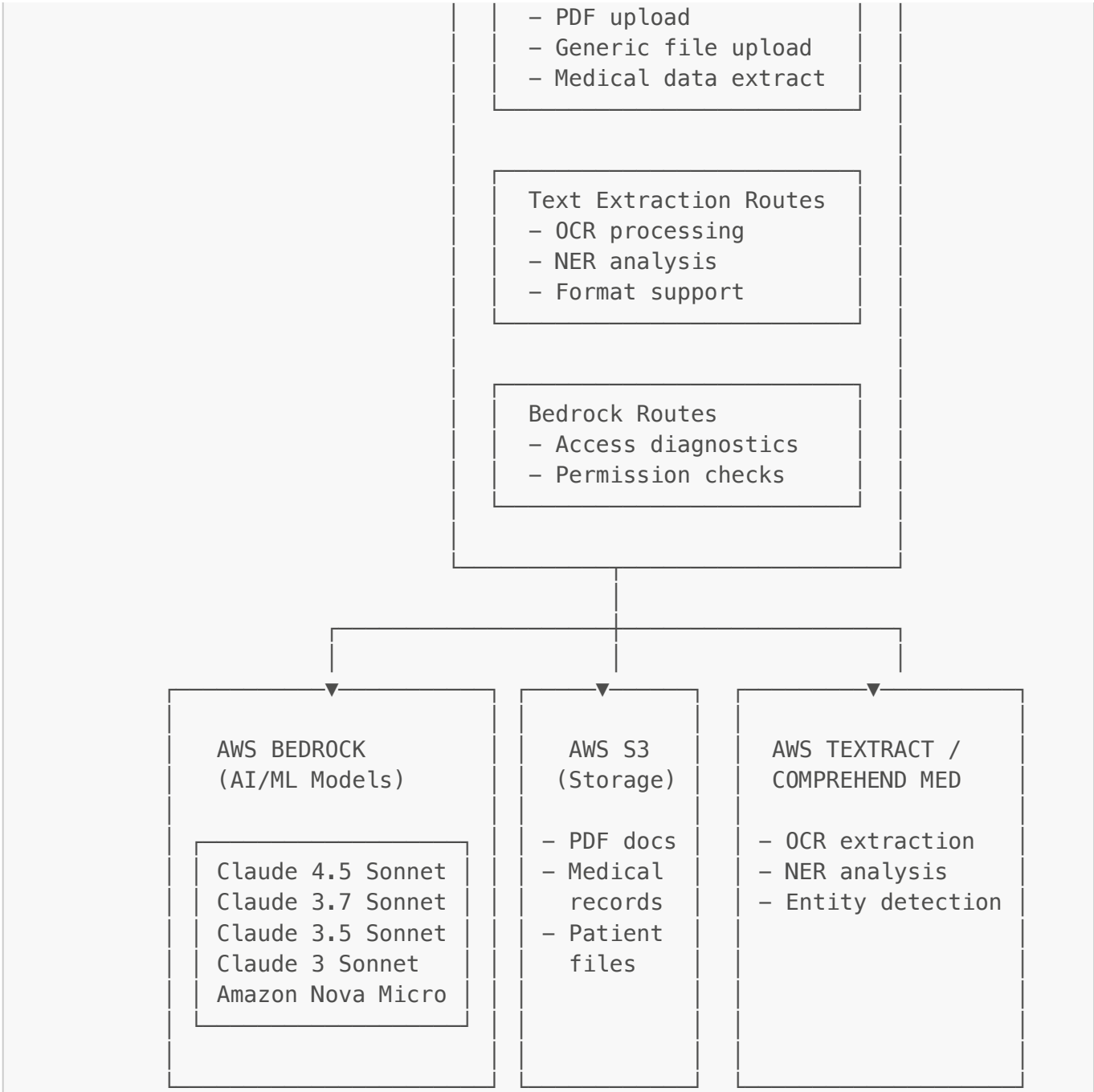
```
[HIGH] Drug Interaction
  Lisinopril + Potassium Chloride: Risk of hyperkalemia with ACE
inhibitors and K+ supplements
  Recommendation: Monitor potassium levels weekly initially

[HIGH] Kidney Function Risk
  Metformin requires dose adjustment or contraindicated in CKD
  Recommendation: Verify GFR and adjust dose accordingly. Consider
alternative if GFR <30.

[MODERATE] Polypharmacy
  Patient prescribed 7 medications - increased risk of adverse events
  Recommendation: Review each medication for continued necessity.
```

---

## Appendix C: Additional System Diagrams

### System Architecture Diagram

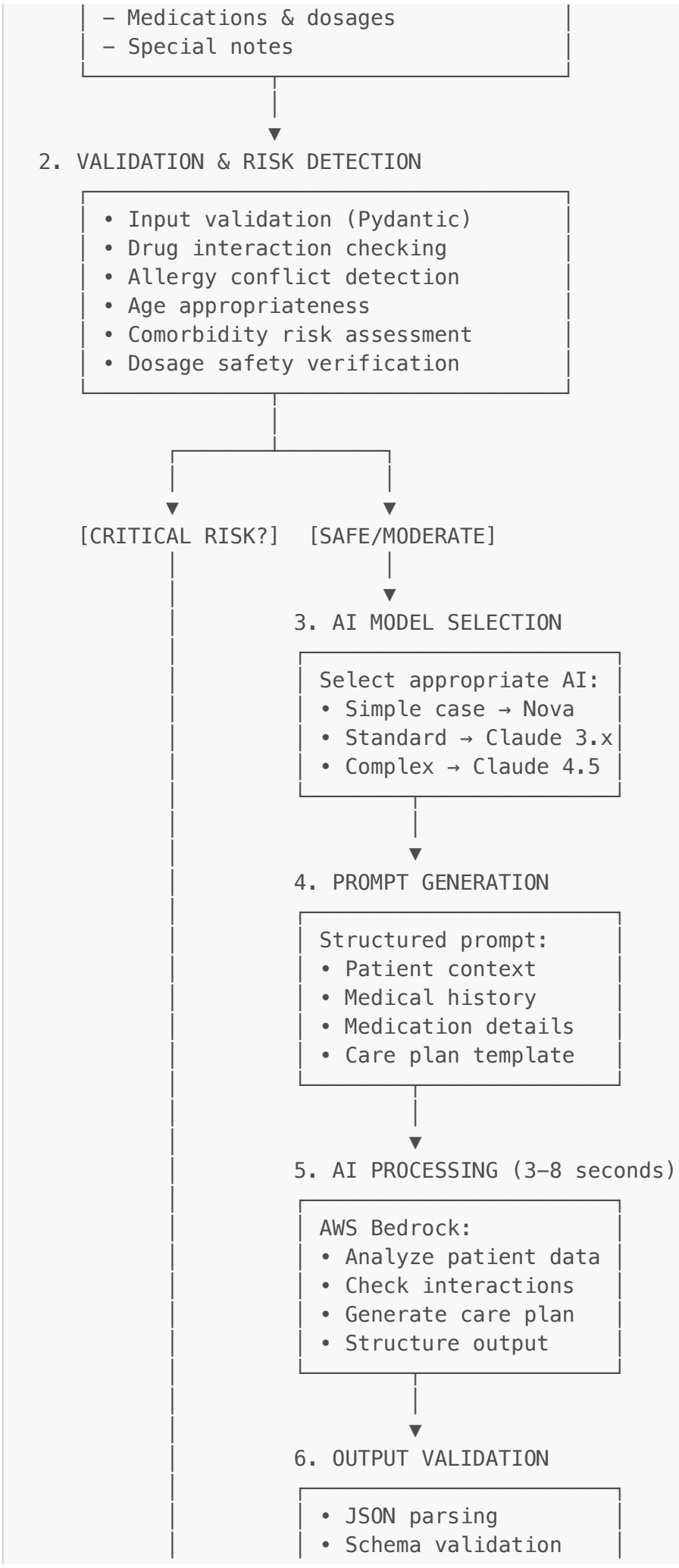


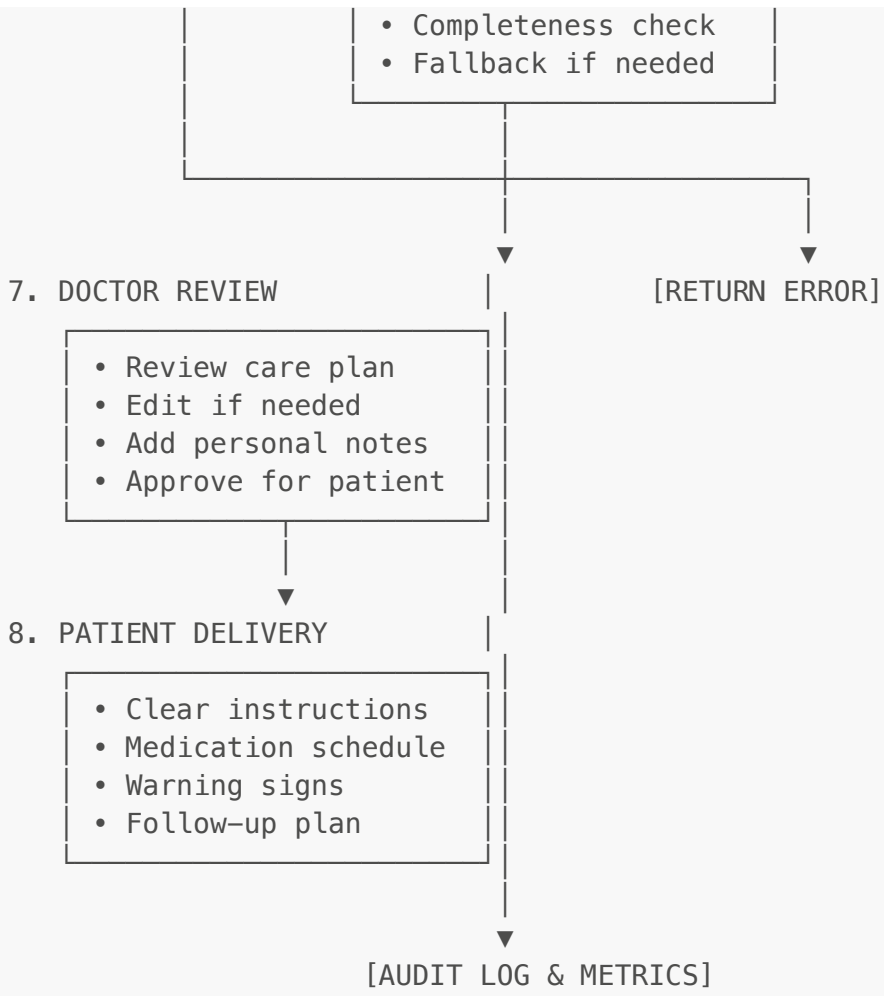



---

### Care Plan Generation Workflow

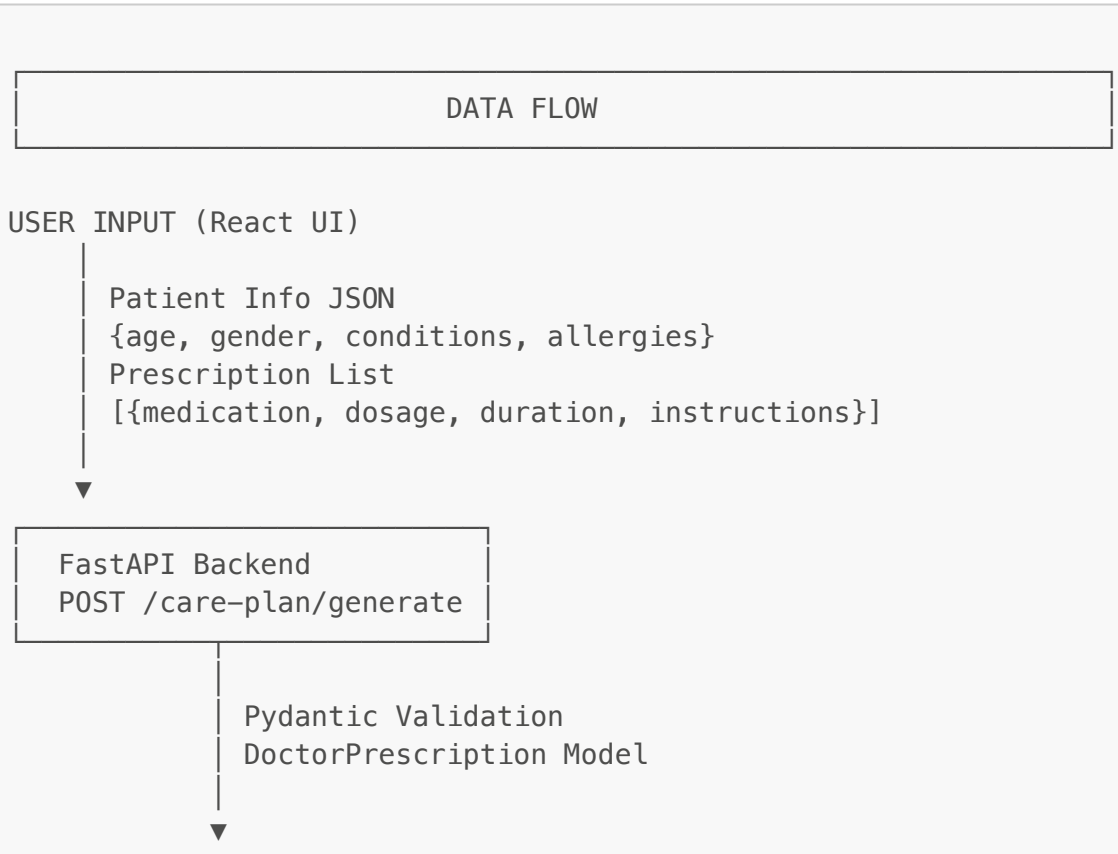
```
┌──────────────────────────────────────────────────────────┐
│                  CARE PLAN GENERATION FLOW               │
└──────────────────────────────────────────────────────────┘

1. INPUT COLLECTION

   ┌──────────────────────────────────────┐
   │ Doctor enters prescription data:     │
   │ - Patient information                │
   │ - Medical conditions                 │
   │ - Allergies                          │
   │ - Diagnosis                          │
```

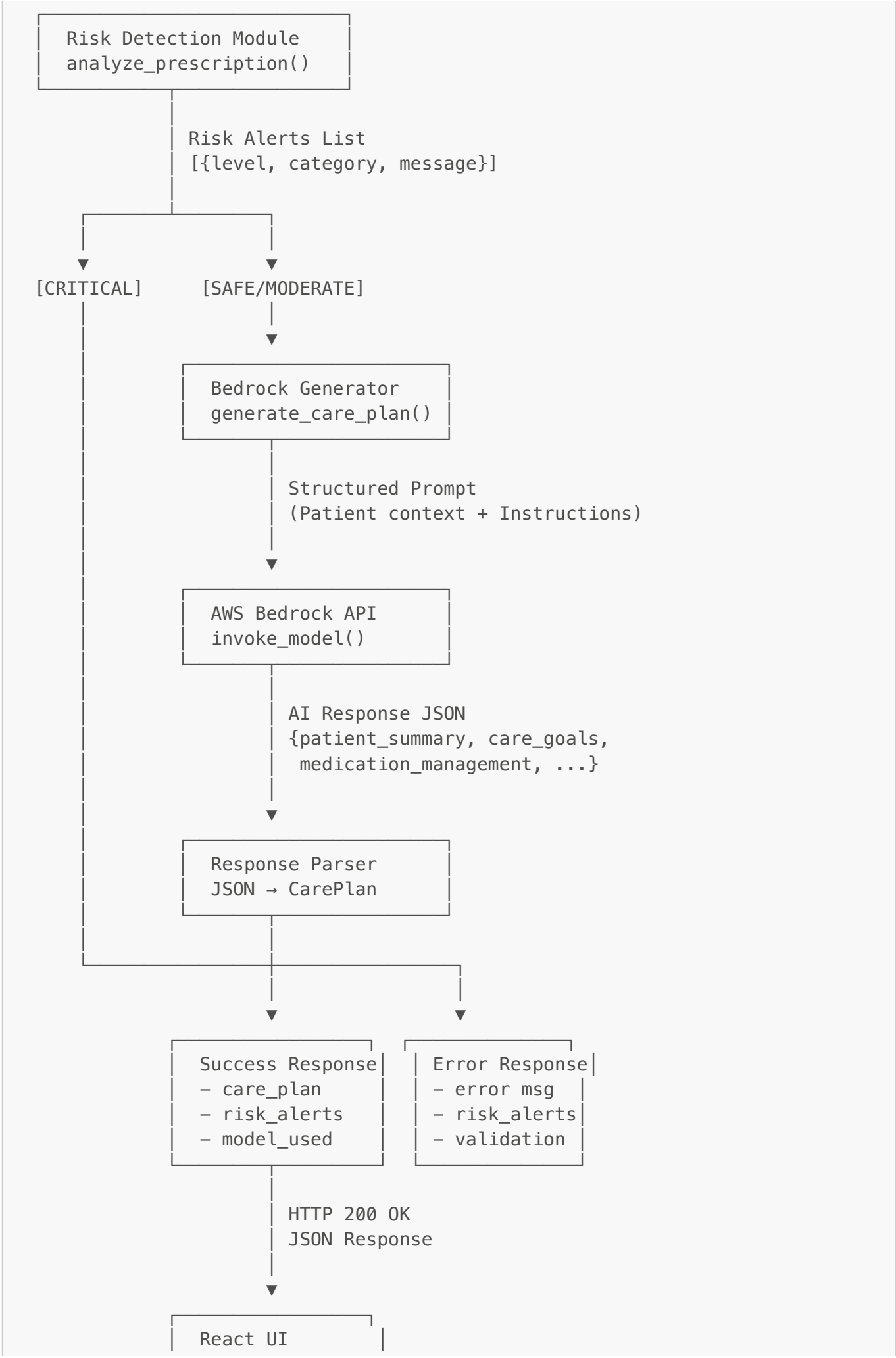

- Medications & dosages
- Special notes
2. VALIDATION & RISK DETECTION
• Input validation (Pydantic)
• Drug interaction checking
• Allergy conflict detection
• Age appropriateness
• Comorbidity risk assessment
• Dosage safety verification
[CRITICAL RISK?]
[SAFE/MODERATE]
3. AI MODEL SELECTION
Select appropriate AI:
• Simple case → Nova
• Standard → Claude 3.x
• Complex → Claude 4.5
4. PROMPT GENERATION
Structured prompt:
• Patient context
• Medical history
• Medication details
• Care plan template
5. AI PROCESSING (3–8 seconds)
AWS Bedrock:
• Analyze patient data
• Check interactions
• Generate care plan
• Structure output
6. OUTPUT VALIDATION
• JSON parsing
• Schema validation

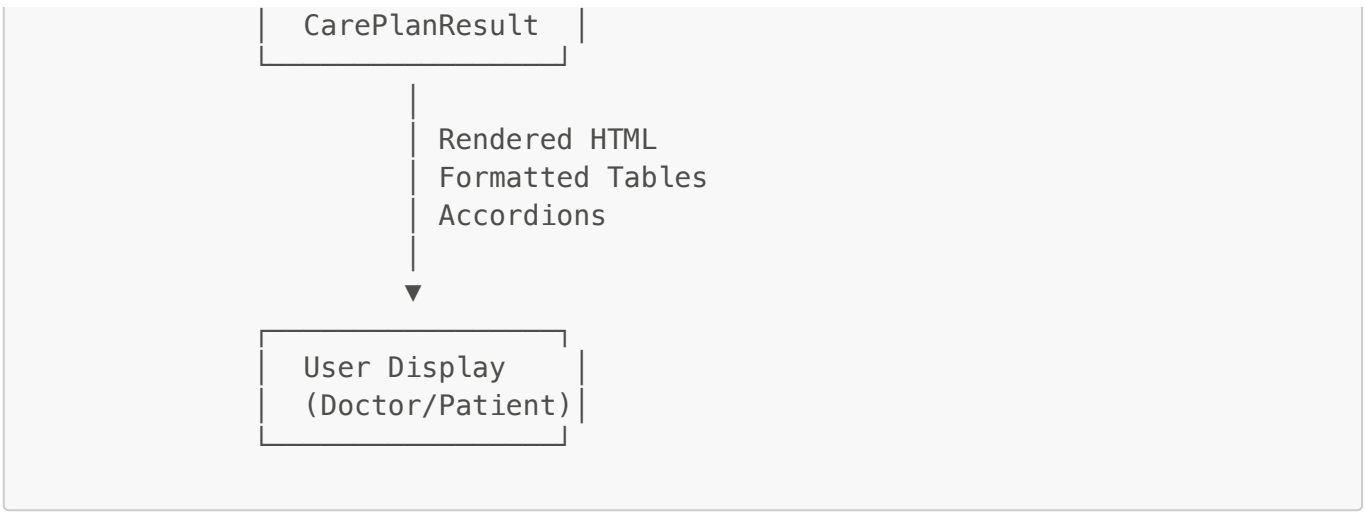



---

## Data Flow Diagram

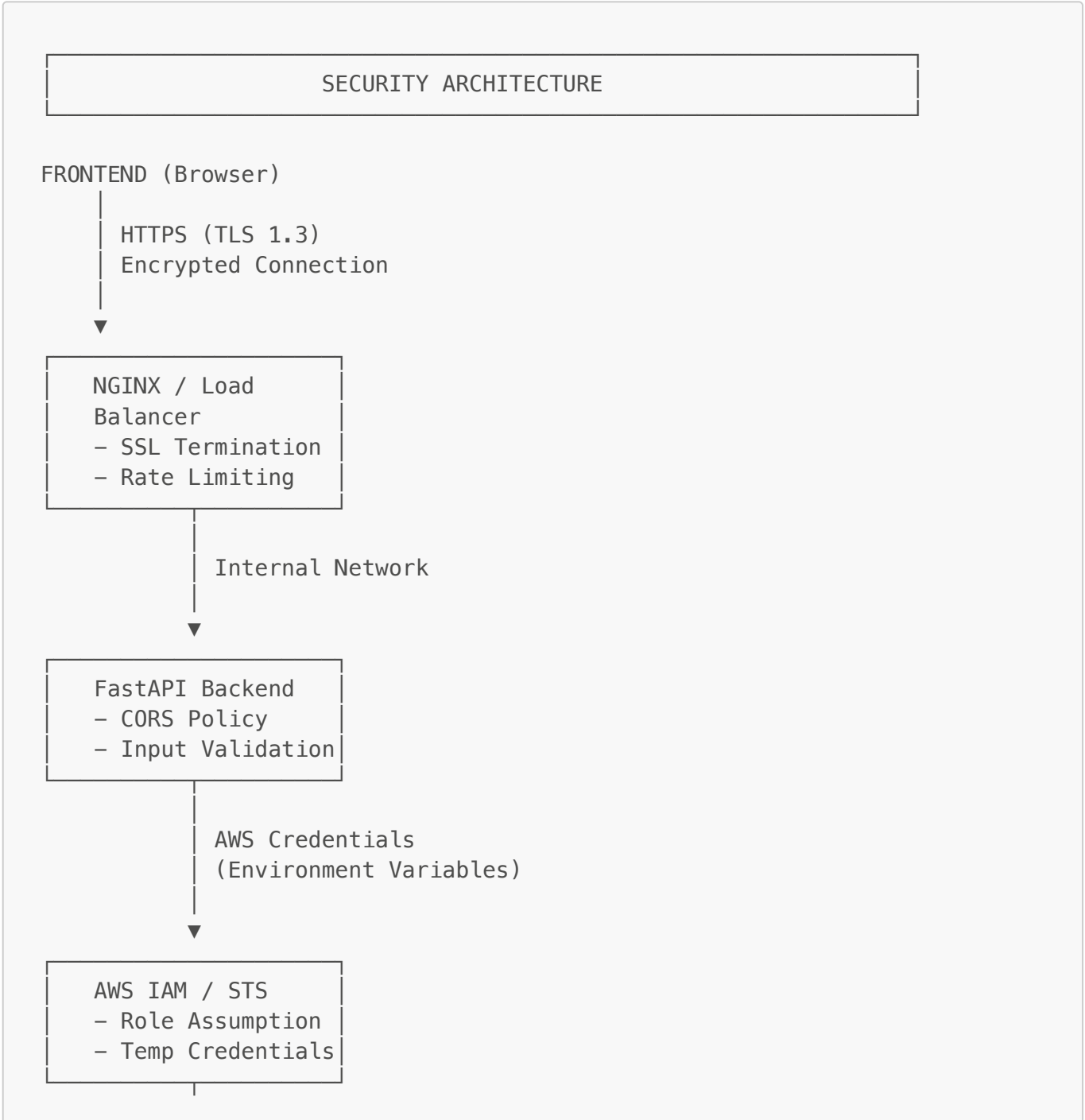


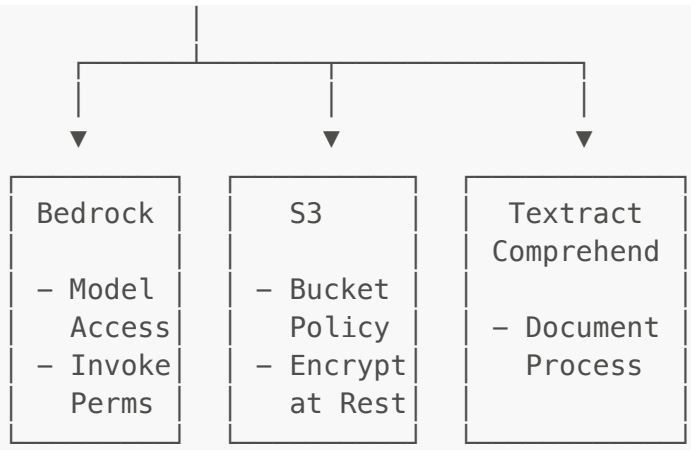

Risk Detection Module
analyze_prescription()
Risk Alerts List
[{level, category, message}]
[CRITICAL]
[SAFE/MODERATE]
Bedrock Generator
generate_care_plan()
Structured Prompt
(Patient context + Instructions)
AWS Bedrock API
invoke_model()
AI Response JSON
{patient_summary, care_goals,
medication_management, ...}
Response Parser
JSON → CarePlan
Success Response
- care_plan
- risk_alerts
- model_used
Error Response
- error msg
- risk_alerts
- validation
HTTP 200 OK
JSON Response
React UI




---

### Security & Authentication Flow

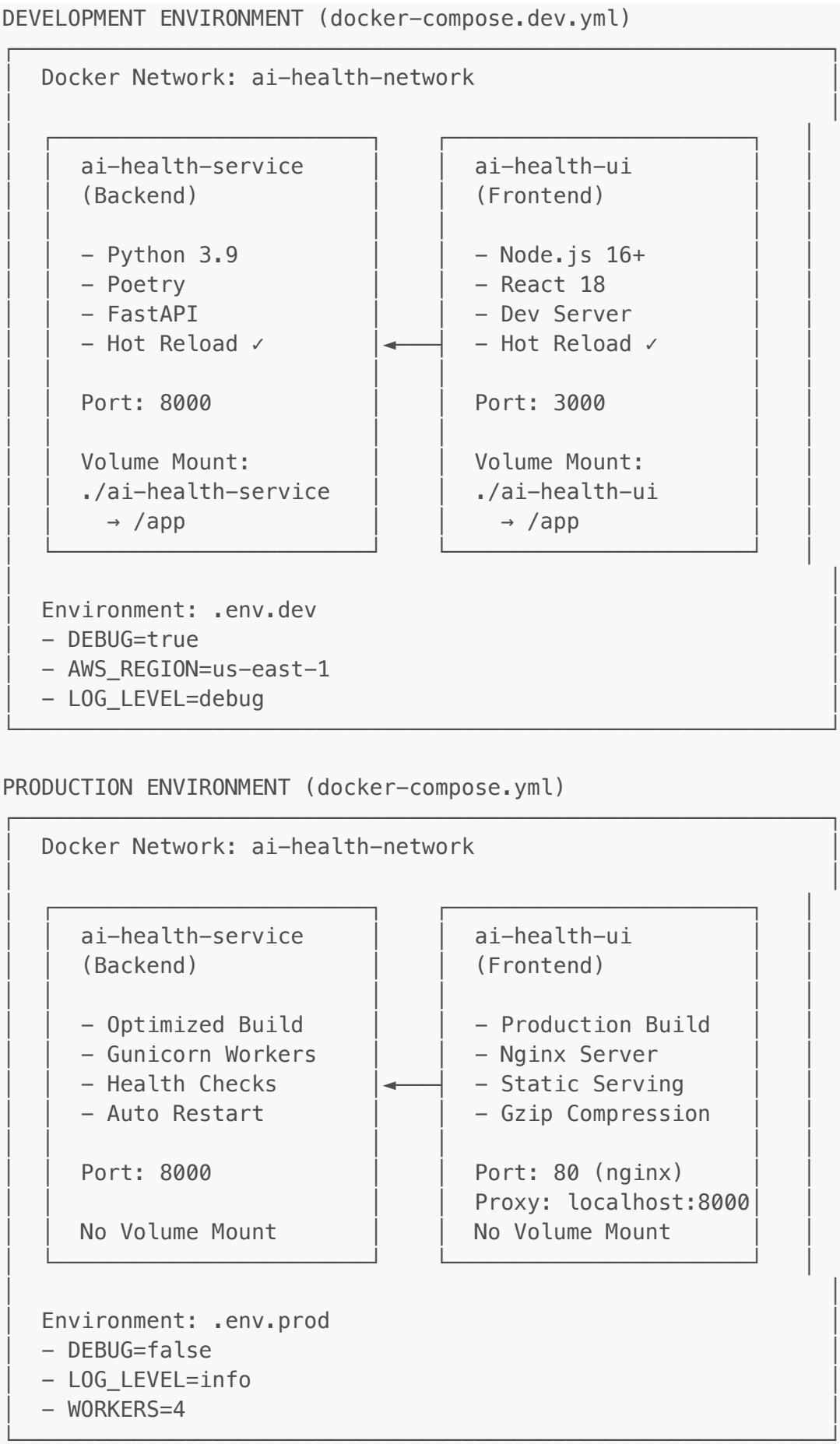





```
SECURITY CONTROLS:
├─ Network Layer
│  ├─ HTTPS only (no HTTP)
│  ├─ TLS 1.3 encryption
│  └─ IP whitelisting (optional)
│
├─ Application Layer
│  ├─ CORS policy enforcement
│  ├─ Input validation (Pydantic)
│  ├─ SQL injection prevention
│  ├─ XSS protection
│  └─ Rate limiting
│
├─ Authentication & Authorization
│  ├─ AWS IAM roles
│  ├─ Principle of least privilege
│  ├─ Temporary credentials (STS)
│  └─ Role-based access control
│
├─ Data Protection
│  ├─ Encryption in transit (TLS)
│  ├─ Encryption at rest (S3, EBS)
│  ├─ No PHI logging
│  └─ HIPAA compliance
│
└─ Audit & Monitoring
   ├─ CloudWatch logging
   ├─ CloudTrail audit logs
   ├─ Access logging
   └─ Error tracking
```

---

### Deployment Architecture


DEVELOPMENT ENVIRONMENT (docker-compose.dev.yml)
Docker Network: ai-health-network
ai-health-service
(Backend)
- Python 3.9
- Poetry
- FastAPI
- Hot Reload ✓
Port: 8000
Volume Mount:
./ai-health-service
→ /app
ai-health-ui
(Frontend)
- Node.js 16+
- React 18
- Dev Server
- Hot Reload ✓
Port: 3000
Volume Mount:
./ai-health-ui
→ /app
Environment: .env.dev
- DEBUG=true
- AWS_REGION=us-east-1
- LOG_LEVEL=debug
PRODUCTION ENVIRONMENT (docker-compose.yml)
Docker Network: ai-health-network
ai-health-service
(Backend)
- Optimized Build
- Gunicorn Workers
- Health Checks
- Auto Restart
Port: 8000
No Volume Mount
ai-health-ui
(Frontend)
- Production Build
- Nginx Server
- Static Serving
- Gzip Compression
Port: 80 (nginx)
Proxy: localhost:8000
No Volume Mount
Environment: .env.prod
- DEBUG=false
- LOG_LEVEL=info
- WORKERS=4
SCALING OPTIONS:
├─ Horizontal Scaling
│ ├─ Multiple backend containers
│ ├─ Load balancer (Nginx/HAProxy)

```
│  └─ Shared state (Redis/Database)
│
├─ Cloud Deployment
│  ├─ AWS ECS / EKS
│  ├─ Auto-scaling groups
│  ├─ Application Load Balancer
│  └─ Multi-AZ deployment
│
└─ Monitoring
   ├─ Container health checks
   ├─ Prometheus metrics
   ├─ Grafana dashboards
   └─ CloudWatch integration
```

---

## Related Documents

- PROJECT_OVERVIEW.md - Complete project details
- CARE_PLAN_BUSINESS_ANALYSIS.md - Business case and analysis
- README.md - Getting started guide
- DEVELOPMENT.md - Development setup
- ARCHITECTURE_SUMMARY.md - Technical architecture

---

*Last Updated: November 8, 2025*
*Document Version: 1.0*
*Part of: AI-Health Platform Documentation*