TRD

Technical Requirements Document (TRD)

AI Car Variant Comparison System

Version: 1.0

Date: January 2026

Document Type: Technical Specification for Development

---

1. SYSTEM OVERVIEW

1.1 Architecture Diagram

```
┌─────────────────────────────────────────────────────────────
────────┐
│                   USER INTERFACE                    │
│                   (Streamlit App)                   │
│                                                     │
│  [Make Dropdown] [Model Dropdown] [Variant Dropdown]   │
│             [Show Recommendations Button]             │
└───────────────────────────┬─────────────────────────────────
────────┘
                            │
                            ▼
┌─────────────────────────────────────────────────────────────
────────┐
│             AGENT ORCHESTRATION LAYER                 │
│                (LangChain + Gemini)                   │
│                                                     │
│  Logic: Receive variant → Query DB → Find upgrades      │
│         → Calculate deltas → Format response            │
└───────────────────────────┬─────────────────────────────────
```

```
────────┘
                    │
                    ▼
┌──────────────────────────────────────────────────────────────────
────────┐
│              TOOL LAYER                    │
│                                      │
│  Tool 1: get_variant_details(make, model, variant)     │
│  Tool 2: find_upgrade_options(current_variant)         │
│  Tool 3: calculate_feature_diff(var1, var2)           │
└────────────────────────────┬─────────────────────────────────────
────────┘
                    │
                    ▼
┌──────────────────────────────────────────────────────────────────
────────┐
│              DATA LAYER                    │
│         (MongoDB Atlas / ChromaDB)               │
│                                      │
│  Collections: car_variants                    │
│  Indexes: make, model, variant_name, variant_tier       │
│  Embeddings: 384-dim vectors for semantic search       │
└──────────────────────────────────────────────────────────────────
────────┘
```

2. DATA MODEL SPECIFICATION

2.1 Database Choice Decision Matrix

Criteria	MongoDB Atlas	ChromaDB (Local)
Setup Time	15 mins	5 mins
Free Tier	512MB, 3 indexes	Unlimited local
Vector Search	Native support	Native support
Internet Dependency	Required	None

Criteria	MongoDB Atlas	ChromaDB (Local)
Hackathon Risk	Low (proven)	Low (simpler)
Production Ready	Yes	Needs migration

DECISION: Start with ChromaDB for hackathon speed, migrate to MongoDB Atlas for production

---

2.2 Collection Schema: `car_variants`

```
{
    # Identification
    "id": "maruti_swift_zxi_plus_2024",
    "make": "Maruti",
    "model": "Swift",
    "variant_name": "ZXi+",
    "variant_tier": "top",  # Values: base, mid, high, top
    "model_year": 2024,

    # Pricing
    "price_ex_showroom": 849000,
    "price_on_road_delhi": 965000,

    # Features (Categorized)
    "features": {
        "safety": [
            "6 airbags",
            "ABS with EBD",
            "ESP (Electronic Stability Program)",
            "Hill Hold Assist",
            "Rear parking sensors",
            "Reverse camera"
        ],
        "comfort": [
            "Automatic climate control",
            "Rear AC vents",
            "Cruise control",
            "Height adjustable driver seat"
        ],
        "technology": [
            "7-inch touchscreen",
            "Apple CarPlay",
            "Android Auto",
            "Wireless charger",
            "Bluetooth connectivity"
        ],
        "exterior": [
            "Electric sunroof",
            "15-inch alloy wheels",
            "LED projector headlamps",
            "LED DRLs",
            "Body colored ORVMs"
        ],
        "convenience": [
            "Keyless entry",
            "Push button start",
            "Auto headlamps",
            "Electrically adjustable ORVMs"
        ]
    },

    # Technical Specifications
    "specifications": {
        "engine_cc": 1197,
        "horsepower": 89,
        "torque_nm": 113,
        "transmission": "5-speed manual",
        "fuel_type": "Petrol",
        "mileage_kmpl": 22.38,
        "fuel_tank_liters": 37,
        "seating_capacity": 5
    },
    # Hierarchy Navigation
    "hierarchy": {
        "tier_order": 4,  # 1=base, 2=mid, 3=high, 4=top
        "previous_variant": "ZXi",
        "next_variant": null  # null for top variant
    },

    # Embeddings for Vector Search
    "feature_embedding": [0.234, -0.456, 0.789, ...],  # 384 dimensions

    # Feature Summary (for embedding generation)
    "feature_summary": "Premium Swift variant with electric sunroof, automatic climate control, 6 airbags, cruise control, wireless charging, LED projector headlamps, 15-inch alloy wheels, keyless entry",

    # Metadata
    "created_at": "2024-01-10T10:30:00Z",
    "data_source": "kaggle_indian_cars_2024"
}
```

2.3 Data Hierarchy Structure

```
Maruti (make)
├── Swift (model)
│   ├── LXi (base) - tier_order: 1
│   ├── VXi (mid) - tier_order: 2
│   ├── ZXi (high) - tier_order: 3
│   └── ZXi+ (top) - tier_order: 4
│
├── Baleno (model)
│   ├── Sigma (base) - tier_order: 1
│   ├── Delta (mid) - tier_order: 2
│   ├── Zeta (high) - tier_order: 3
│   └── Alpha (top) - tier_order: 4
```

```
│
└── Brezza (model)
    ├── LXi (base) - tier_order: 1
    ├── VXi (mid) - tier_order: 2
    ├── ZXi (high) - tier_order: 3
    └── ZXi+ (top) - tier_order: 4
```

---

3. AGENT DESIGN SPECIFICATION

3.1 Agent Architecture

Framework: LangChain (v0.1.0+)

LLM: Google Gemini Pro (gemini-pro)

Pattern: Tool-based agent (not ReAct for simplicity)

3.2 Agent Tools Definition

Tool 1: `get_variant_details`

Purpose: Fetch complete details of user-selected variant

Function Signature:

```
def get_variant_details(make: str, model: str, variant_name: str) → dict:
    """
    Retrieves variant information from database.

    Args:
        make: Car brand (e.g., "Maruti")
        model: Car model (e.g., "Swift")
        variant_name: Variant name (e.g., "VXi")

    Returns:
        {
            "variant_name": "VXi",
            "price": 699000,
            "features": {...},
            "tier_order": 2
        }
    """
```

Implementation Logic:

1. Query database with filters: `make`, `model`, `variant_name`
2. Return single document
3. Handle not found error → return error message

Tool 2: `find_upgrade_options`

Purpose: Find next 2 higher tier variants in same model

Function Signature:

```
def find_upgrade_options(make: str, model: str, current_tier: int) → list:
    """
    Finds upgrade variants (higher tiers only).

    Args:
        make: Car brand
        model: Car model
        current_tier: Current variant's tier_order

    Returns:
        [
            {
                "variant_name": "ZXi",
                "price": 799000,
                "tier_order": 3,
                "features": {...}
            },
            {
                "variant_name": "ZXi+",
                "price": 849000,
                "tier_order": 4,
                "features": {...}
            }
        ]
    """
```

Implementation Logic:

1. Query: `make=X AND model=Y AND tier_order > current_tier`
2. Sort by `tier_order` ascending
3. Limit to 2 results
4. If results < 2, return available (handles top variant case)

Tool 3: `calculate_feature_difference`

Purpose: Compare two variants and extract unique features

Function Signature:

```
def calculate_feature_difference(variant1: dict, variant2: dict) → dict:
    """
    Calculates price delta and unique features.

    Args:
        variant1: Lower tier variant data
        variant2: Higher tier variant data

    Returns:
        {
            "price_difference": 100000,
            "additional_features": [
                "Sunroof",
                "2 extra airbags",
                "Cruise control"
            ],
            "cost_per_feature": 33333,
            "feature_categories": {
                "safety": ["2 extra airbags"],
                "comfort": ["Cruise control"],
                "exterior": ["Sunroof"]
            }
        }
    """
```

Implementation Logic:

1. Calculate: `price_diff = variant2['price'] - variant1['price']`
2. For each feature category (safety, comfort, tech, exterior, convenience):
    - Find features in variant2 NOT in variant1 (set difference)
3. Flatten all unique features into single list
4. Calculate: `cost_per_feature = price_diff / len(additional_features)`
5. Return structured diff

3.3 Agent Workflow

Input: User selections (make, model, variant_name)

Execution Steps:

```
# Step 1: Get selected variant
selected = get_variant_details(make, model, variant_name)

# Step 2: Find upgrade options
upgrades = find_upgrade_options(make, model, selected['tier_order'])

# Step 3: If no upgrades (top variant)
if len(upgrades) == 0:
    return "You've selected the top variant! No upgrades available."

# Step 4: Calculate differences for each upgrade
```

```
recommendations = []
for upgrade in upgrades:
    diff = calculate_feature_difference(selected, upgrade)
    recommendations.append({
        "variant": upgrade,
        "diff": diff
    })

# Step 5: Format response
return format_recommendations(selected, recommendations)
```

Output Format:

```
{
    "selected_variant": {
        "name": "VXi",
        "price": 699000,
        "features": [...]
    },
    "upgrade_options": [
        {
            "variant_name": "ZXi",
            "price": 799000,
            "price_increase": 100000,
            "additional_features": ["Auto AC", "Alloy wheels"],
            "cost_per_feature": 50000,
            "recommendation": "Good value upgrade"
        },
        {
            "variant_name": "ZXi+",
            "price": 849000,
            "price_increase": 150000,
            "additional_features": ["Sunroof", "6 airbags", "Cruise control"],
            "cost_per_feature": 50000,
            "recommendation": "Premium choice"
        }
```

```
  ]
}
```

4. EMBEDDING STRATEGY

4.1 Why Embeddings?

Use Case: Enable semantic search for similar variants across brands

- Example: "Find variants similar to Swift ZXi+ in other brands"
- Future feature (not hackathon MVP)

4.2 Embedding Model

Model: SentenceTransformer (`all-MiniLM-L6-v2`)

- **Dimensions:** 384
- **Runs:** Locally (no API cost)
- **Speed:** ~50ms per embedding

4.3 Embedding Generation Process

Input Text Construction:

```
def create_embedding_text(variant_doc):
    text = f"""
    {variant_doc['make']} {variant_doc['model']} {variant_doc['variant_name']}
    Price: {variant_doc['price_ex_showroom']} rupees
    Tier: {variant_doc['variant_tier']}
    Safety: {', '.join(variant_doc['features']['safety'])}
    Comfort: {', '.join(variant_doc['features']['comfort'])}
    Technology: {', '.join(variant_doc['features']['technology'])}
    Exterior: {', '.join(variant_doc['features']['exterior'])}
    """
    return text.strip()
```

Embedding Generation:

```
from sentence_transformers import SentenceTransformer

embedder = SentenceTransformer('all-MiniLM-L6-v2')

for variant in variants:
    text = create_embedding_text(variant)
    embedding = embedder.encode(text).tolist()  # Returns 384-dim list
    variant['feature_embedding'] = embedding
    # Save to database
```

5. UI SPECIFICATION (STREAMLIT)

5.1 Page Layout

```
┌──────────────────────────────────────────────────────────────
─┐
│  🚗 AI Car Variant Advisor                    │
│  Find the perfect variant for your budget          │
└──────────────────────────────────────────────────────────────
─┘

┌──────────────────────────────────────────────────────────────
──┐
│  SELECT YOUR CAR                              │
│                                      │
│  Brand:    [Dropdown: Maruti ▼]                 │
│  Model:    [Dropdown: Swift ▼]                  │
│  Variant:  [Dropdown: VXi ▼]                    │
│                                      │
│           [Show Recommendations]                │
└──────────────────────────────────────────────────────────────
──┘

┌──────────────────────────────────────────────────────────────
```

```
─┐
│ YOUR SELECTION                          │
│                                         │
│ Maruti Swift VXi                        │
│ ₹6,99,000 (Ex-showroom)                 │
│                                         │
│ Features:                               │
│ Safety: 4 airbags, ABS, EBD             │
│ Comfort: Manual AC, Fabric seats        │
│ Technology: Basic touchscreen, Bluetooth │
│ Exterior: Steel wheels, Halogen headlamps │
└──────────────────────────────────────────
─┘

┌──────────────────────────────────────────
─┐
│ 🤖 SMART UPGRADE SUGGESTIONS            │
└──────────────────────────────────────────
─┘

┌──────────────────────────────────────────
─┐
│ OPTION 1: Swift ZXi                     │
│ ₹7,99,000 (+₹1,00,000)                  │
│                                         │
│ Additional Features:                    │
│ ✓ Automatic climate control             │
│ ✓ 15-inch alloy wheels                  │
│ ✓ Rear parking sensors                  │
│ ✓ LED DRLs                              │
│                                         │
│ 💡 Value: ₹25,000 per feature           │
│ [Consider This Upgrade]                 │
└──────────────────────────────────────────
─┘
```

```
┌─────────────────────────────────────────────
──┐
│ OPTION 2: Swift ZXi+                        │
│ ₹8,49,000 (+₹1,50,000)                      │
│                                         │
│ Additional Features:                        │
│ ✓ Electric sunroof                          │
│ ✓ 6 airbags (2 extra)                       │
│ ✓ Cruise control                            │
│ ✓ Wireless charger                          │
│ ✓ All ZXi features                          │
│                                         │
│ 💡 Value: ₹30,000 per feature               │
│ [Premium Choice]                            │
└─────────────────────────────────────────────
──┘
```

5.2 UI Components Specification

Component 1: Selection Panel

- 3 cascading dropdowns
- Make dropdown → Triggers model dropdown population
- Model dropdown → Triggers variant dropdown population
- Submit button with loading state

Component 2: Selected Variant Card

- Large heading with variant name
- Price prominently displayed
- Features grouped by category
- Expandable sections for each category

Component 3: Upgrade Suggestion Cards

- Maximum 2 cards

- Highlighted price difference (green text)
- Feature list with checkmarks
- Cost-per-feature calculation
- Call-to-action button (future: links to dealer)

Component 4: Top Variant Handler

- Special message card
- Trophy/star icon
- "You've selected the best!" message
- List all features (no comparison needed)

6. API CONTRACTS

6.1 Database Query API

Query 1: Get All Makes

```
GET /api/makes
Response: ["Maruti", "Hyundai", "Tata", "Mahindra", "Honda"]
```

Query 2: Get Models by Make

```
GET /api/models?make=Maruti
Response: ["Swift", "Baleno", "Brezza", "Ertiga"]
```

Query 3: Get Variants by Model

```
GET /api/variants?make=Maruti&model=Swift
Response: [
  {"name": "LXi", "tier": "base"},
  {"name": "VXi", "tier": "mid"},
  {"name": "ZXi", "tier": "high"},
  {"name": "ZXi+", "tier": "top"}
]
```

Query 4: Get Full Variant Details

```
GET /api/variant?make=Maruti&model=Swift&variant=VXi
Response: {
  "id": "maruti_swift_vxi_2024",
  "variant_name": "VXi",
  "price": 699000,
  "features": {...},
  "tier_order": 2
}
```

6.2 Agent API

Endpoint: `/agent/recommend`

Request:

```
{
  "make": "Maruti",
  "model": "Swift",
  "variant_name": "VXi"
}
```

Response:

```
{
  "status": "success",
  "selected_variant": {
    "name": "VXi",
    "price": 699000,
    "features": {
      "safety": ["4 airbags", "ABS"],
      "comfort": ["Manual AC"],
      "technology": ["Basic touchscreen"],
      "exterior": ["Steel wheels"]
    }
  },
  "upgrade_options": [
    {
      "variant_name": "ZXi",
      "price": 799000,
      "price_increase": 100000,
      "additional_features": {
          "comfort": ["Auto AC"],
          "exterior": ["Alloy wheels"],
          "safety": ["Rear sensors"]
      },
      "feature_count": 3,
      "cost_per_feature": 33333
    },
    {
      "variant_name": "ZXi+",
      "price": 849000,
      "price_increase": 150000,
      "additional_features": {
          "exterior": ["Sunroof"],
          "safety": ["2 extra airbags"],
          "comfort": ["Cruise control"],
          "technology": ["Wireless charger"]
      },
      "feature_count": 5,
      "cost_per_feature": 30000
    }
  ],
  "is_top_variant": false
}
```

Response (Top Variant Case):

```
{
    "status": "success",
    "selected_variant": {
        "name": "ZXi+",
        "price": 849000,
        "features": {...}
    },
    "upgrade_options": [],
    "is_top_variant": true,
    "message": "You've selected the top variant with all available features!"
}
```

7. DATA PIPELINE

7.1 Data Collection Process

Step 1: Download Kaggle Dataset

```
kaggle datasets download -d medhekarabhinav5/indian-cars-dataset
unzip indian-cars-dataset.zip
```

Step 2: Data Cleaning Script

```
import pandas as pd

# Load raw data
df = pd.read_csv('indian_cars_raw.csv')

# Clean column names
df.columns = df.columns.str.lower().str.replace(' ', '_')

# Extract hierarchy
df['variant_tier'] = df['variant_name'].apply(assign_tier)
df['tier_order'] = df['variant_tier'].map({
    'base': 1, 'mid': 2, 'high': 3, 'top': 4
})

# Parse features from description column
df['features'] = df['description'].apply(parse_features)

# Save cleaned data
df.to_json('variants_clean.json', orient='records', indent=2)
```

Step 3: Feature Parsing Logic

```
def parse_features(description_text):
    """
    Extract features from unstructured text.
    Example input: "6 airbags, sunroof, cruise control, alloy wheels"
    """
    features = {
        "safety": [],
        "comfort": [],
        "technology": [],
        "exterior": [],
        "convenience": []
    }

    # Feature keyword mapping
    safety_keywords = ['airbag', 'abs', 'ebd', 'esp', 'sensor', 'camera']
    comfort_keywords = ['ac', 'climate', 'seat', 'cruise']
    tech_keywords = ['touchscreen', 'carplay', 'android', 'bluetooth', 'charger']
    exterior_keywords = ['sunroof', 'alloy', 'led', 'drl', 'wheel']

    # Parse and categorize
    for item in description_text.lower().split(','):
        item = item.strip()
        if any(kw in item for kw in safety_keywords):
            features['safety'].append(item)
        elif any(kw in item for kw in comfort_keywords):
            features['comfort'].append(item)
        # ... continue for other categories

    return features
```

7.2 Database Ingestion

ChromaDB Setup:

```
import chromadb
from chromadb.utils import embedding_functions

# Initialize ChromaDB
client = chromadb.PersistentClient(path="./car_variants_db")

# Create collection with embedding function
sentence_transformer_ef = embedding_functions.SentenceTransformerEmbeddingFunction(
    model_name="all-MiniLM-L6-v2"
)

collection = client.create_collection(
    name="car_variants",
    embedding_function=sentence_transformer_ef,
    metadata={"description": "Car variant specifications"}
)

# Load cleaned data
import json
with open('variants_clean.json', 'r') as f:
    variants = json.load(f)

# Insert documents
ids = [v['id'] for v in variants]
documents = [create_embedding_text(v) for v in variants]
```

```
metadatas = variants  # Store full variant data as metadata

collection.add(
    ids=ids,
    documents=documents,
    metadatas=metadatas
)
```

8. TECHNOLOGY DEPENDENCIES

8.1 Python Requirements

```
# requirements.txt

# Core Framework
streamlit==1.30.0
langchain==0.1.0
langchain-google-genai==0.0.6

# Database
chromadb==0.4.22
# OR
pymongo==4.6.1  # If using MongoDB Atlas

# Embeddings
sentence-transformers==2.2.2

# Data Processing
pandas==2.1.4
numpy==1.24.3

# Utilities
python-dotenv==1.0.0
requests==2.31.0
```

8.2 Environment Variables

```
# API Keys
GEMINI_API_KEY=your_gemini_api_key_here

# Database (if MongoDB Atlas)
MONGODB_URI=mongodb+srv://username:password@cluster.mongodb.net/

# Application
DEBUG_MODE=True
LOG_LEVEL=INFO
```

9. DEVELOPMENT WORKFLOW

9.1 Project Structure

```
car-variant-advisor/
│
├── data/
│   ├── raw/
│   │   └── indian_cars_raw.csv
│   ├── processed/
│   │   └── variants_clean.json
│   └── embeddings/
│       └── car_variants_db/  # ChromaDB storage
│
├── src/
│   ├── __init__.py
│   ├── database/
│   │   ├── __init__.py
│   │   ├── chroma_client.py
│   │   └── queries.py
│   │
│   ├── agent/
│   │   ├── __init__.py
│   │   ├── tools.py
│   │   └── orchestrator.py
│   │
│   └── utils/
│       ├── __init__.py
│       ├── data_loader.py
│       └── feature_parser.py
│
├── app/
│   └── streamlit_app.py
│
├── scripts/
│   ├── 01_download_data.py
│   ├── 02_clean_data.py
│   ├── 03_generate_embeddings.py
│   └── 04_ingest_to_db.py
│
├── tests/
│   ├── test_tools.py
│   ├── test_agent.py
│   └── test_database.py
│
├── .env
├── .gitignore
├── requirements.txt
└── README.md
```

9.2 Development Phases

Phase 1: Data Setup (Day 1-2)

```
# Step 1: Download and clean data
python scripts/01_download_data.py
python scripts/02_clean_data.py
```

```
# Step 2: Generate embeddings
python scripts/03_generate_embeddings.py

# Step 3: Ingest to database
python scripts/04_ingest_to_db.py

# Verify
python -c "from src.database.queries import get_all_makes; print(get_all_makes())"
```

Phase 2: Tool Development (Day 3-4)

```
# Test individual tools
python -m pytest tests/test_tools.py -v

# Test tool 1
python -c "from src.agent.tools import get_variant_details; print(get_variant_details('Maruti', 'Swift', 'VXi'))"

# Test tool 2
python -c "from src.agent.tools import find_upgrade_options; print(find_upgrade_options('Maruti', 'Swift', 2))"
```

Phase 3: Agent Integration (Day 5)

```
# Test agent orchestrator
python -m pytest tests/test_agent.py -v

# Manual test
python -c "
from src.agent.orchestrator import get_recommendations
result = get_recommendations('Maruti', 'Swift', 'VXi')
print(result)
"
```

Phase 4: UI Development (Day 6)

```
# Run Streamlit app
streamlit run app/streamlit_app.py

# Test in browser at http://localhost:8501
```

Phase 5: Deployment (Day 7)

```
# Deploy to Streamlit Cloud
git add .
git commit -m "Final hackathon submission"
git push origin main

# Configure in Streamlit Cloud dashboard
# Add secrets (GEMINI_API_KEY)
# Deploy from GitHub repo
```

---

10. ERROR HANDLING

10.1 Database Errors

Error: Variant Not Found

```
def get_variant_details(make, model, variant_name):
    try:
        result = collection.get(
            where={"$and": [
                {"make": make},
                {"model": model},
                {"variant_name": variant_name}
            ]}
        )
        if len(result['ids']) == 0:
            return {
                "error": "Variant not found",
                "message": f"No variant '{variant_name}' found for {make} {model}"
            }
        return result['metadatas'][0]
    except Exception as e:
        return {"error": str(e)}
```

Error: Database Connection Failed

```
try:
    client = chromadb.PersistentClient(path="./car_variants_db")
    collection = client.get_collection("car_variants")
except Exception as e:
    st.error(f"Database connection failed: {e}")
    st.stop()
```

10.2 Agent Errors

Error: LLM API Timeout

```
import time
from langchain.llms import GoogleGenerativeAI

def get_llm_with_retry(max_retries=3):
    for attempt in range(max_retries):
        try:
            llm = GoogleGenerativeAI(model="gemini-pro", temperature=0.3)
            # Test call
            llm("test")
            return llm
        except Exception as e:
            if attempt < max_retries - 1:
                time.sleep(2 ** attempt)  # Exponential backoff
            else:
                raise e
```

Error: No Upgrade Options Available

```
def find_upgrade_options(make, model, current_tier):
    results = collection.get(
        where={"$and": [
            {"make": make},
            {"model": model},
            {"tier_order": {"$gt": current_tier}}
        ]},
        limit=2
    )

    if len(results['ids']) == 0:
        return {
            "is_top_variant": True,
            "message": "You've selected the top variant!"
        }

    return results['metadatas']
```

10.3 UI Errors

Error: Empty Dropdown Selection

```
# In Streamlit app
make = st.selectbox("Brand", options=get_all_makes())
if not make:
    st.warning("Please select a brand")
    st.stop()

model = st.selectbox("Model", options=get_models_by_make(make))
if not model:
    st.warning("Please select a model")
    st.stop()
```

Error: API Rate Limit

```
try:
    recommendations = agent.run(query)
except Exception as e:
    if "rate limit" in str(e).lower():
        st.error("Too many requests. Please wait 30 seconds and try again.")
        time.sleep(30)
    else:
        st.error(f"Error: {e}")
```

---

11. TESTING STRATEGY

11.1 Unit Tests

Test: Tool 1 - Get Variant Details

```
def test_get_variant_details():
    result = get_variant_details("Maruti", "Swift", "VXi")
assert result['variant_name'] == "VXi"
assert result['price'] == 699000
assert 'features' in result
```

Test: Tool 2 - Find Upgrades

```
def test_find_upgrade_options():
    # Test mid-tier (should return 2 upgrades)
    result = find_upgrade_options("Maruti", "Swift", 2)
    assert len(result) == 2
    assert result[0]['tier_order'] == 3
    assert result[1]['tier_order'] == 4

    # Test top-tier (should return 0 upgrades)
    result_top = find_upgrade_options("Maruti", "Swift", 4)
    assert result_top['is_top_variant'] == True
```

Test: Tool 3 - Feature Difference

```
def test_calculate_feature_difference():
    vxi = get_variant_details("Maruti", "Swift", "VXi")
    zxi = get_variant_details("Maruti", "Swift", "ZXi")

    diff = calculate_feature_difference(vxi, zxi)

    assert diff['price_difference'] == 100000
    assert len(diff['additional_features']) > 0
    assert diff['cost_per_feature'] > 0
```

11.2 Integration Tests

Test: End-to-End Agent Flow

```
def test_agent_recommendation_flow():
    # Test normal variant
    result = get_recommendations("Maruti", "Swift", "VXi")
    assert result['status'] == 'success'
    assert len(result['upgrade_options']) > 0
    assert result['is_top_variant'] == False

    # Test top variant
    result_top = get_recommendations("Maruti", "Swift", "ZXi+")
    assert result_top['is_top_variant'] == True
    assert len(result_top['upgrade_options']) == 0
```

11.3 Demo Test Cases

Test Case 1: Base Variant Selection

- Select: Maruti → Swift → LXi

- Expected: 2 upgrade suggestions (VXi, ZXi)

Test Case 2: Mid Variant Selection

- Select: Hyundai → Creta → EX
- Expected: 2 upgrade suggestions (S, SX)

Test Case 3: Top Variant Selection

- Select: Tata → Nexon → XZ+ Lux
- Expected: "You've selected top variant" message

Test Case 4: Model with 3 Variants

- Select: Mahindra → Scorpio → S11
- Expected: 1 upgrade suggestion (only top variant available)

12. PERFORMANCE REQUIREMENTS

12.1 Response Time Targets

Operation	Target	Maximum
Dropdown population	< 100ms	200ms
Database query	< 200ms	500ms
Agent recommendation	< 5s	10s
Full page load	< 3s	5s

12.2 Resource Limits

ChromaDB:

- Storage: < 100MB for 50-70 variants
- Memory: < 500MB during operation

Streamlit App:

- Memory: < 1GB
- CPU: Single core sufficient

Gemini API:

- Free tier: 60 requests/minute
- Strategy: Cache common queries

13. DEPLOYMENT SPECIFICATION

13.1 Streamlit Cloud Configuration

File: `.streamlit/config.toml`

```
[theme]
primaryColor="#FF4B4B"
backgroundColor="#FFFFFF"
secondaryBackgroundColor="#F0F2F6"
textColor="#262730"
font="sans serif"

[server]
headless = true
port = 8501
enableCORS = false
```

File: `.streamlit/secrets.toml` (Not committed to Git)

```
GEMINI_API_KEY = "your_api_key_here"
```

13.2 Deployment Steps

Step 1: Prepare Repository

```
# Create .gitignore
echo ".env
*.pyc
__pycache__/
.streamlit/secrets.toml
```

```
data/embeddings/
.DS_Store" > .gitignore

# Commit code
git add .
git commit -m "Ready for deployment"
git push origin main
```

Step 2: Streamlit Cloud Setup

1. Go to share.streamlit.io
2. Connect GitHub account
3. Select repository
4. Set main file: `app/streamlit_app.py`
5. Add secrets (GEMINI_API_KEY)
6. Deploy

Step 3: Post-Deployment Verification

- Test all 3 dropdowns
- Test recommendation flow
- Test top variant edge case
- Check mobile responsiveness

14. MONITORING & LOGGING

14.1 Application Logging

```
import logging

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('app.log'),
        logging.StreamHandler()
    ]
)

logger = logging.getLogger(__name__)

# Usage in tools
def get_variant_details(make, model, variant_name):
    logger.info(f"Fetching variant: {make} {model} {variant_name}")
    try:
        result = collection.get(...)
        logger.info(f"Found variant: {result['id']}")
        return result
    except Exception as e:
        logger.error(f"Error fetching variant: {e}")
        raise
```

14.2 Performance Monitoring

```
import time

def monitor_performance(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()

        logger.info(f"{func.__name__} took {end_time - start_time:.2f}s")
        return result
    return wrapper

@monitor_performance
```

```
def get_variant_details(make, model, variant_name):
    # ... function code
```

15. SECURITY CONSIDERATIONS

15.1 API Key Management

Never commit `.env` or secrets.toml

```
# In .gitignore
.env
.streamlit/secrets.toml
*.key
```

Use environment variables

```
import os
from dotenv import load_dotenv

load_dotenv()

GEMINI_API_KEY = os.getenv('GEMINI_API_KEY')
if not GEMINI_API_KEY:
    raise ValueError("GEMINI_API_KEY not found in environment")
```

15.2 Input Validation

```
def validate_input(make, model, variant_name):
    # Whitelist validation
    valid_makes = get_all_makes()
    if make not in valid_makes:
        raise ValueError(f"Invalid make: {make}")

    # SQL injection prevention (not applicable for ChromaDB, but good practic
e)
```

```
    if any(char in variant_name for char in [';', '--', '/*']):
        raise ValueError("Invalid characters in variant name")

    return True
```

16. FUTURE ENHANCEMENTS ROADMAP

Phase 1: Post-Hackathon (Month 1)

- Add user context input ("I have 2 kids")
- Prioritize features based on context
- Add feature comparison table view

Phase 2: Production Ready (Month 2-3)

- Scale to 50+ models across 10 brands
- Add real-time price updates (API integration)
- User authentication and saved comparisons

Phase 3: Advanced Features (Month 4-6)

- Cross-model comparison (Swift vs i20)
- EMI calculator integration
- Resale value prediction (add XGBoost here)
- Test drive booking

17. SUCCESS CRITERIA

17.1 Hackathon Demo Success

Must Achieve:

- ✅ System works for 5 models without errors
- ✅ Recommendations are accurate (100% match expected)

- ✅ UI loads in < 5 seconds
- ✅ Agent responds in < 10 seconds
- ✅ Demo video recorded and submitted

17.2 Technical Quality Metrics

- ✅ Code coverage > 70% (unit tests)
- ✅ No hardcoded data in application code
- ✅ Clean separation: UI → Agent → Database
- ✅ Proper error handling (no crashes)
- ✅ README with clear setup instructions

17.3 Judge Appeal Factors

- ✅ Clean, professional UI
- ✅ Clear value proposition ("pay X, get Y")
- ✅ Works end-to-end in live demo
- ✅ Scalable architecture shown
- ✅ Future roadmap articulated

18. RISK MITIGATION

Risk	Probability	Impact	Mitigation
Dataset missing variants	Medium	High	Manually add 20 key variants
Gemini API quota exceeded	Low	High	Implement response caching
ChromaDB corruption	Low	Medium	Daily backups, version control
Streamlit Cloud deployment fails	Low	High	Test locally first, have video backup

Risk	Probability	Impact	Mitigation
Agent gives wrong suggestions	Medium	High	Unit tests for all edge cases

19. APPENDIX

19.1 Sample Data Document

```
{
    "id": "maruti_swift_zxi_plus_2024",
    "make": "Maruti",
    "model": "Swift",
    "variant_name": "ZXi+",
    "variant_tier": "top",
    "model_year": 2024,
    "price_ex_showroom": 849000,
    "price_on_road_delhi": 965000,
    "features": {
        "safety": [
            "6 airbags",
            "ABS with EBD",
            "ESP",
            "Hill Hold Assist",
            "Rear parking sensors",
            "Reverse camera"
        ],
        "comfort": [
            "Automatic climate control",
            "Rear AC vents",
            "Cruise control",
            "Height adjustable driver seat"
        ],
        "technology": [
            "7-inch touchscreen",
            "Apple CarPlay",
            "Android Auto",
            "Wireless charger",
            "Bluetooth 5.0"
        ],
        "exterior": [
            "Electric sunroof",
            "15-inch alloy wheels",
            "LED projector headlamps",
            "LED DRLs",
            "Body colored ORVMs with turn indicators"
        ],
        "convenience": [
            "Keyless entry",
            "Push button start",
            "Auto headlamps",
            "Rain sensing wipers",
            "Electrically adjustable ORVMs"
        ]
    },
    "specifications": {
        "engine_cc": 1197,
        "horsepower": 89,
        "torque_nm": 113,
        "transmission": "5-speed manual",
        "fuel_type": "Petrol",
        "mileage_kmpl": 22.38,
        "fuel_tank_liters": 37,
        "seating_capacity": 5,
        "boot_space_liters": 265
    },
    "hierarchy": {
        "tier_order": 4,
        "previous_variant": "ZXi",
        "next_variant": null
    },
    "feature_embedding": [0.234, -0.456, 0.789, ...],
    "feature_summary": "Premium Swift variant with electric sunroof, automatic climate control, 6 airbags, cruise control, wireless charging, LED projector headlamps, 15-inch alloy wheels, keyless entry, push button start",
    "created_at": "2024-01-10T10:30:00Z",
    "data_source": "kaggle_indian_cars_2024"
}
```

19.2 Tool Implementation Reference

Complete Tool 1 Implementation:

```
from typing import Dict, Optional
import chromadb

client = chromadb.PersistentClient(path="./car_variants_db")
collection = client.get_collection("car_variants")

def get_variant_details(make: str, model: str, variant_name: str) → Dict:
    """
    Retrieves complete variant information from database.

    Args:
        make: Car manufacturer (e.g., "Maruti")
        model: Car model (e.g., "Swift")
        variant_name: Variant trim name (e.g., "VXi")

    Returns:
        Dictionary with variant details or error message
    """
    try:
        results = collection.get(
            where={
                "$and": [
                    {"make": make},
                    {"model": model},
                    {"variant_name": variant_name}
                ]
            }
        )

        if len(results['ids']) == 0:
            return {
                "error": "NOT_FOUND",
                "message": f"Variant '{variant_name}' not found for {make} {model}"
            }

        # Return first matching document (should be unique)
        variant_data = results['metadatas'][0]

        return {
            "status": "success",
            "data": variant_data
        }

    except Exception as e:
        logger.error(f"Database error in get_variant_details: {e}")
        return {
            "error": "DATABASE_ERROR",
            "message": str(e)
        }
```

20. FINAL CHECKLIST

Before Code Implementation

- [✓] PRD approved
- [✓] TRD reviewed and understood
- [✓] Data source confirmed (Kaggle dataset accessible)

- [✓] API keys obtained (Gemini)
- [✓] Development environment set up

Before Demo

□ All 5 models tested

□ Edge cases verified (top variant, single upgrade)

□ UI polished and responsive

□ Demo script prepared

□ Video recorded (backup)

□ GitHub repo cleaned and documented

Before Submission

□ README.md complete with setup instructions

□ requirements.txt verified

□ .env.example provided (without real keys)

□ Code commented appropriately

□ Presentation slides ready

Document Status: ✅ Ready for Implementation

Next Step: Begin Phase 1 - Data Setup

Proceed to coding?