Credit Card Rewards Optimizer: Singapore Edition

A Singapore-focused application that uses Generative AI to analyze consumer spending patterns and provide multi-card synergy recommendations with strategic usage guidelines. The system uses an agentic architecture leveraging the Model Context Protocol (MCP) to power AI-driven chat for scenario planning and T&C clarifications.

Features

- **Spending Analysis**: Analyze transaction data to create a comprehensive spending profile
- **Merchant Categorization**: Categorize merchant names into spending categories using a distilled model trained on ACRA business data
- **Card Recommendation**: Recommend optimal credit card combinations based on spending patterns and user preferences
- **Strategic Usage**: Provide specific advice on which card to use for which spending category
- **T&C Insights**: Answer natural language questions about card terms and conditions

Project Structure

```
project/
├── src/                       # Main source code
│   ├── app.py                 # Streamlit application UI
│   ├── main.py                # Main entry point for demo
│   ├── agent.py               # Card optimizer agent implementation
│   ├── mcp/                   # MCP implementation
│   │   ├── client.py          # MCP client for server communication
│   │   └── card_data_server.py  # Card data access server
│   ├── models/                # AI models
│   │   ├── merchant_categorizer.py  # Merchant categorization model
│   │   ├── pdf_statement_parser.py  # PDF parsing module
│   │   └── vector_db.py       # Vector database implementation
│   └── mcp_servers/           # MCP server implementations
│       └── card_data_server.py  # Card data access server
├── data/                      # Data storage
│   ├── card_tcs/              # Card terms and conditions
│   │   └── pdf/               # Original T&C PDFs
│   ├── sample_statements/     # Sample card statements for testing
│   └── vector_db/             # Vector database storage
├── run_app.py                 # Entry point for Streamlit app
├── requirements.txt           # Python dependencies
└── docs/                      # Documentation files
```

Setup Instructions

1. Create a virtual environment:

```
python -m venv venv
source venv/bin/activate  # On Windows: .\venv\Scripts\Activate.ps1
```

2. Install requirements:

```
pip install -r requirements.txt
```

3. Create a .env file by copying .env.example and filling in required values.

4. Start the MCP server:

```
python -m src.model_context_protocol.card_data_server
```

5. Run the Streamlit application (in a new terminal window):

```
python run_app.py
```

Component Overview

1. Transaction Categorization

- `src/models/merchant_categorizer.py` - Distilled model for categorizing merchant names

2. Card Embeddings & Semantic Search

- `src/card_processing/vector_db.py` - Vector database for card embeddings
- `src/mcp/card_data_server.py` - MCP tools for card data access including semantic search

3. Agent Reasoning

- `src/agent.py` - Main agent implementation for card recommendations and scenario analysis

4. RAG Pipeline

- `src/card_processing/vector_db.py` - Vector database for T&C document storage
- `src/mcp/card_data_server.py` - Tools for T&C querying

5. UI & DevOps

- `src/app.py` - Streamlit UI implementation
- `run_app.py` - Application entry point
- `Dockerfile` and `docker-compose.yml` - Container configuration

MCP Tool Architecture

The system implements four core MCP tools as defined in the PRD:

1. `get_available_cards()` - Returns a list of all available cards with basic metadata
2. `get_card_details(card_id)` - Returns complete card information in its original format
3. `query_tc(question, card_id)` - Natural language queries about card terms and conditions
4. `search_cards(query)` - Semantic search for cards matching natural language criteria

Tasks For Team Members

Team Member 1: Transaction Data Wrangling & Merchant Categorization

- Implement `merchant_categorizer.py` with proper model distillation
- Complete `pdf_statement_parser.py` for statement parsing
- Test categorization with real Singapore merchant examples

Team Member 2: Card Embeddings & Semantic Search

- Implement vector embedding functionality in `vector_db.py` for cards
- Update `search_cards` in `card_data_server.py` to use vector search
- Collect and structure Singsaver card data

Team Member 3: Agent Reasoning & Strategy Implementation

- Complete `agent.py` with actual LLM-based reasoning
- Implement multi-card synergy calculations
- Create advanced scenario analysis

Team Member 4: RAG Pipeline & Question Suggestions

- Complete T&C document ingestion with `vector_db.py`
- Improve `query_tc` function with proper RAG retrieval
- Implement dynamic suggested questions generator

Team Member 5: DevOps, MCP & Chat Orchestration

- Complete `app.py` Streamlit interface
- Implement multi-step conversation flow in chat
- Finalize Docker configuration for deployment

Testing

Each component should include its own `if __name__ == "__main__"` block for quick testing.

Run the complete flow demo with:

```
python src/main.py
```

Future Work

- Add support for additional banks and card issuers
- Implement foreign transaction fee analysis
- Add support for statement upload via API
- Enhance UI with visualizations of rewards optimization

License