AI AGENT-POWERED AUTOMATION FOR PELOTON FITNESS ECOSYSTEM:
DESIGN & PROTOTYPE PHASE

Kevin Geidel
MSDS 442: AI Agent Design & Development
Northwestern University
June 1, 2025

Requirement 1: Directory structure and training data

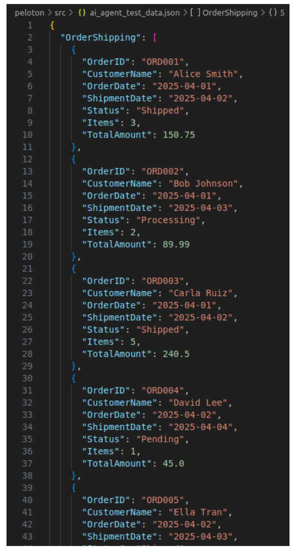
There are a number of files involved in prototyping the Peloton automation AI agent. A full breakdown of the project's code base can be found in figure 1. `Analysis.pdf` is this document. `requirements.txt` is a list of all the Python packages needed to run the agent. They can be installed using `pip` and a Python virtual environment manager (such as `pyenv`.) There are some instructions for cloning the repository in order to run the agent on your local machine in the `README`.

```
msds442/
├── Analysis.pdf
├── requirements.txt
├── README.md
└── peloton/
    ├── src/
    │   ├── ai_agent_test_data.json
    │   └── peloton_data_chatgpt_raw_responses.ipynb
    ├── __init__.py
    ├── agents.py
    └── Phase_2_NOTEBOOK_Geidel.ipynb
```

Figure 1: Directory structure for Peloton AI Agent

Inside the `peloton/` directory we see the `src/` directory. This contains the agent data used for testing. The test data was generated (in phase 1) through a series of prompts to ChatGPT (`https://chatgpt.com/`). The prompts and raw outputs are stored in `peloton_data_chatgpt_raw_responses.ipynb`. The generated test data covers the five AI agents (marketing, data science, membership & fraud detection, orders and product recommendations.) Each agent has test data that supports at least three specific user stories. The records are stored, in JSON format, in `ai_agent_test_data.json`. A sample of what these JSON objects look like can be seen in figure 2

Back in the `peloton/` directory we see `__init__.py`. This file makes the `peloton/` directory recognizable as a Python module to the interpreter and allows us to import it as such. The actual `PelotonAgent` class is defined in `agents.py` and will be described in further detail below (requirement 3.) The Jupyter notebook file, `Phase_2_NOTEBOOK_Geidel.ipynb`, is used for the demonstration. It simply imports the agent class, instantiates an instance, renders a graphic that depicts the LangGraph and invokes the agent.



peloton > src > {} ai_agent_test_data.json > [ ] OrderShipping > {} 5

```json
{
  "OrderShipping": [
    {
      "OrderID": "ORD001",
      "CustomerName": "Alice Smith",
      "OrderDate": "2025-04-01",
      "ShipmentDate": "2025-04-02",
      "Status": "Shipped",
      "Items": 3,
      "TotalAmount": 150.75
    },
    {
      "OrderID": "ORD002",
      "CustomerName": "Bob Johnson",
      "OrderDate": "2025-04-01",
      "ShipmentDate": "2025-04-03",
      "Status": "Processing",
      "Items": 2,
      "TotalAmount": 89.99
    },
    {
      "OrderID": "ORD003",
      "CustomerName": "Carla Ruiz",
      "OrderDate": "2025-04-01",
      "ShipmentDate": "2025-04-02",
      "Status": "Shipped",
      "Items": 5,
      "TotalAmount": 240.5
    },
    {
      "OrderID": "ORD004",
      "CustomerName": "David Lee",
      "OrderDate": "2025-04-02",
      "ShipmentDate": "2025-04-04",
      "Status": "Pending",
      "Items": 1,
      "TotalAmount": 45.0
    },
    {
      "OrderID": "ORD005",
      "CustomerName": "Ella Tran",
      "OrderDate": "2025-04-02",
      "ShipmentDate": "2025-04-03",
```


Figure 2: A sample of the test data in JSON format.

Requirement 2: LangGraph architecture

The actual agent graph is created (using `LangGraph`) in the `build_graph` method of the `PelotonAgent` class (see lines 116-143 in the attached listing.) The nodes and their respective functions are defined above this. Note the use of conditional edges to allow for agent choice and the inclusion of a `ToolNode` that houses the LangChain retriever. The result of integrating these agents into this workflow can be seen graphically in figure 3. Dotted lines indicate conditional edges.

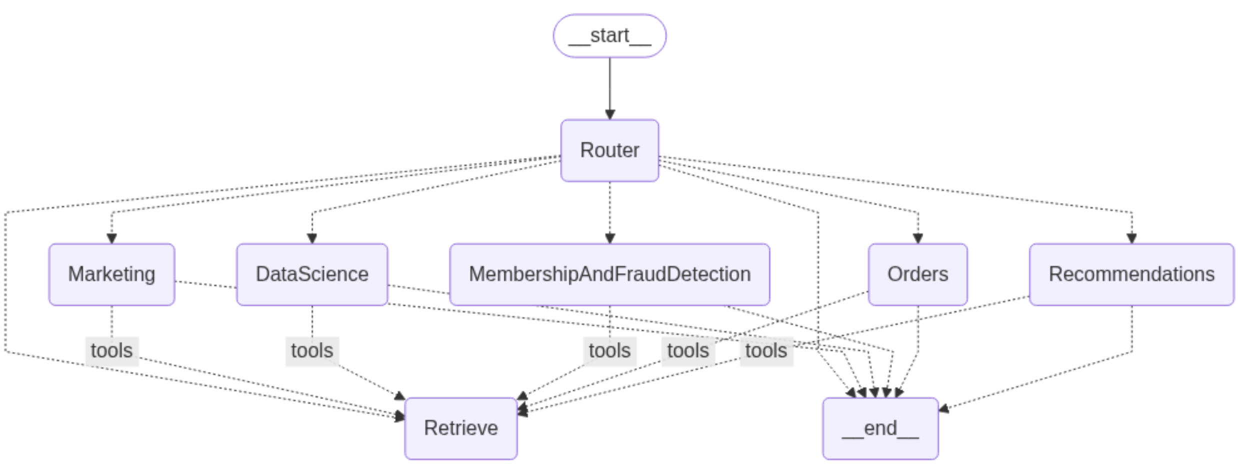


Figure 3: The Peloton agent architecture.

Requirement 3: Implementation

Three area specific agents (plus the router agent) are implemented in this phase (marketing, data science and membership/fraud detection.) In order to discuss their implementation we must examine the `PelotonAgent` class defined in `agents.py`. The contents of this file are included in the attached listing. `LangGraph` is a stateful machine and we give our state attributes in lines 47-52. Each node will track the current inquiry, the latest response, the node that referred the inquiry to the current node and the name of the next node to forward to. Messages are included to provide memory in the form of historical context. Upon instantiating an object of this class we create an LLM client, load the test data into LangChain documents and store these documents in a ChromaDb vector store (lines 61, 64 and 67 respectively.) `load_documents` and `populate_vector_store` and class methods defined below. There are several additional worker methods defined on the class that perform utility functions- mostly to abstract/DRY-out the code. The agent definitions themselves begin on line 201 with the `router_agent`. The core function of the router is to classify the user's intent and set `next_node` accordingly.

The implemented agents share similar patterns. They are centered around a system prompt that defines their purpose and queries the LLM on behalf of the user. All messages, AI, System and Human are added to the messages list to maintain a memory of the conversation over multiple iterations.

Requirement 4: Working prototype

Three user stories (one for each of the implemented agents) were tested. All three agents are able to access their respective data from the store. The marketing agent was developed so that it might extract analyze the performance of ongoing marketing campaigns. The data science agent is able to derive insights from metrics in the data store. The membership agent can answer questions about potential fraudulent accounts. The following transcript of a session tried to demonstrate all of the implemented features: routing, accessing data, drawing insights and access to general knowledge from LLM training.

```
User:
  Hello?
Agent:
  Hello there, this is the Peloton automated AI agent. How can I assist you today?
User:
  Can you recommend a good product?
Agent:
  recommendation_agent is not yet implemented.
User:
  What is the highest performing marketing campaign?
Agent:
  The highest performing marketing campaign is the "Black Friday" campaign, which
has a Click Rate of 7.1 and a Conversion Rate of 3.5.
User:
  When does the next one launch?
Agent:
  The next marketing campaign, "Summer Blast," launches on June 1, 2025.
User:
  What is the workout completion rate for new users?
Agent:
  The workout completion rate for new users is 72.5%.
User:
  Which user segment had the highest LTV?
Agent:
  The user segment with the highest LTV (Lifetime Value) is "Premium Members,"
with a predicted LTV of 1400.
User:
  What factors typically determine LTV?
Agent:
  Factors that typically determine Lifetime Value (LTV) include:  1. **Average
Purchase Value**: The average amount a customer spends per transaction. 2.
**Purchase Frequency**: How often a customer makes a purchase within a specific
time frame. 3. **Customer Lifespan**: The average duration a customer continues
to make purchases from a business. 4. **Churn Rate**: The rate at which
customers stop doing business with a company. 5. **Gross Margin**: The
difference between sales and the cost of goods sold, which affects
profitability. 6. **Customer Acquisition Cost (CAC)**: The cost associated with
acquiring a new customer, which impacts overall profitability. 7. **Engagement
Metrics**: Customer engagement levels, such as interaction with marketing
campaigns or product usage, can influence repeat purchases.  These factors
collectively help businesses estimate the total revenue a customer is expected
to generate throughout their relationship with the company.
User:
  How many login issues did user U002 report?
Agent:
  User U002 reported 3 login issues.
User:
  Which membership types are more likely to have fradulent accounts?
Agent:
  Membership types that are more likely to have fraudulent accounts include:  1.
**Free Memberships**: Users with free memberships have reported fraudulent
activity, with user U002 and user U005 both flagged for fraud. 2. **Trial
Memberships**: While there is only one user in this category (U003), they have
not been flagged for fraud, indicating that this type may be less likely to have
fraudulent accounts compared to free memberships.  In summary, free memberships
appear to be more susceptible to fraudulent accounts based on the reported
issues.
User:
  That's all for me, goodbye!
Agent:
  Goodbye! If you have any more questions in the future, feel free to ask. Have a
great day!
User:
  q
Goodbye!
```

Figure 4: Example session that jogs three user stories and other features.