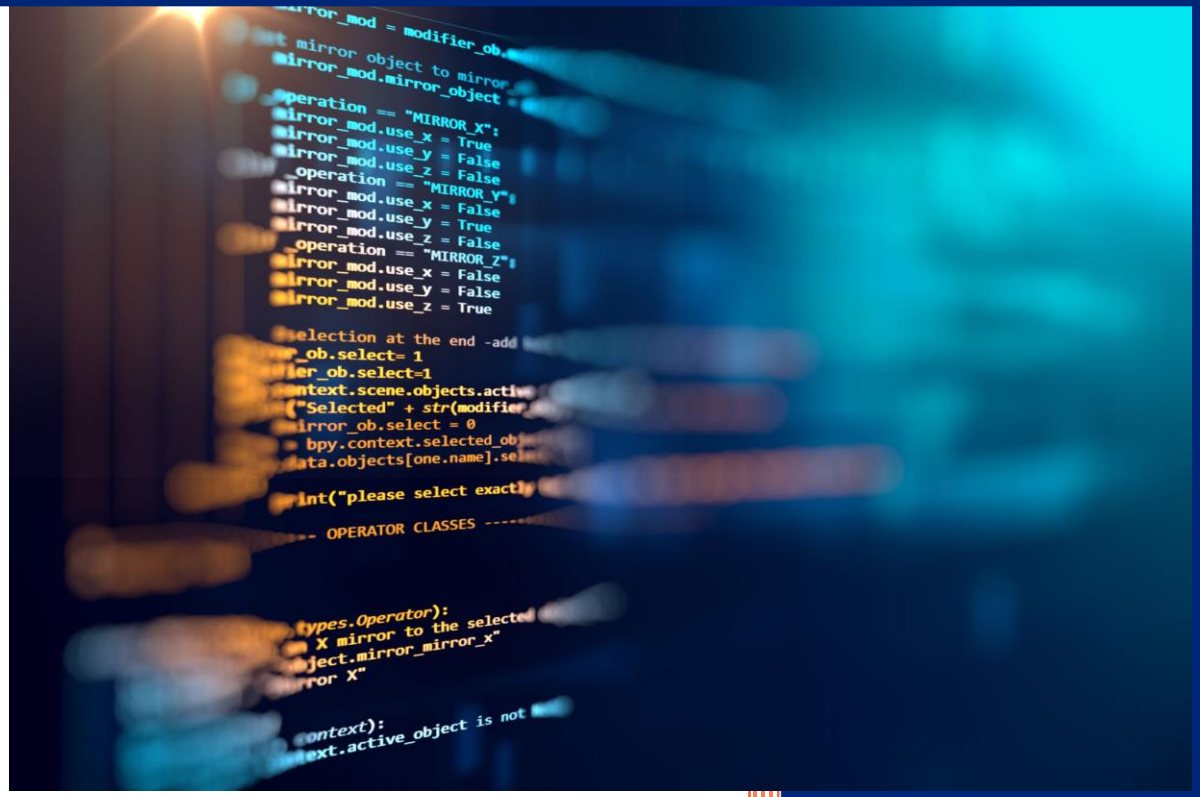
Optum

Instructions For Capstone



United Health Group

AI DOJO

Purpose of the Document

This document is your guide to successfully complete the capstone project. This capstone project is a graded assignment and is the final step towards your GenAI certification.

Kindly read the document carefully before you start your work on the capstone. Additionally, you can download the document and keep it for reference.

In case you have any additional questions along the way, feel free to drop us an email at the following email IDs:

1. UAIS Queries
2. General Content/Capstone related queries for AI Dojo

Table of Contents

The Capstone: Healthcare Insurance Claim Approval Agent

Project Outcomes

In this assignment, you will apply key skills gained during the Generative AI Track, including:

- Apply Generative AI techniques to solve complex real-world challenges
- Build domain-specific AI applications for targeted business problems
- Apply Agentic AI techniques to complex, real-world business processes
- Build domain-specific Agentic AI applications that require specialized knowledge
- Implement and optimize ReAct and Tool-Use Agent architectures for decision-critical applications

Project Overview

The Healthcare Insurance Claim Approval Agent capstone project challenges you to build a robust, intelligent insurance claim validation system using cutting-edge Agentic AI techniques and frameworks such as LangChain and LangGraph.

This project marks the culmination of your applied Agentic AI training and focuses on automating insurance claim coverage determination in a domain where precision, compliance, and reliability are critical.

You will develop an AI-powered agent system that can:

- Interpret semi-structured patient health records along with submitted insurance claims
- Perform retrieval and reasoning with relevant insurance policy guidelines

- Automate the insurance claim coverage decision-making process by generating contextually accurate and well-justified outcomes, including auto-approvals or routing for manual review

By bridging the gap between raw clinical and claim data and actionable insurance outcomes, this project demonstrates the practical, real-world impact of Agentic AI in healthcare insurance administration.

The Challenge

Modern healthcare insurance claim processing faces significant challenges in efficiency, accuracy, and scalability. The large volume and complexity of medical claims, each requiring nuanced interpretation of patient context and detailed policy rules, make manual review methods both time-consuming and inconsistent.

This capstone addresses these challenges by enabling you to build an intelligent system that can:

- Parse and summarize semi-structured patient records and claim details using domain knowledge
- Retrieve and evaluate relevant insurance policy guidelines for specific claims
- Determine claim coverage by matching claim details to policy rules and requirements **(diagnoses, procedures, age, gender and preauthorization)**
- Generate clear, contextually accurate coverage decisions and provide reasoning for approvals or further routing for manual reviews

By completing this capstone, you will demonstrate your ability to design and implement end-to-end Agentic AI systems that automate high-stakes decision-making in healthcare insurance. This will help deliver both operational efficiency and consistent, policy-compliant outcomes.

The Assignment

Your goal is to create an Advanced Agentic AI Claim Coverage Validation System.

This intelligent system would be able to:

- Parse and summarize semi-structured patient records and claim details using domain knowledge
- Retrieve and evaluate relevant insurance policy guidelines for specific claims
- Determine claim coverage by matching claim details to policy rules and requirements **(diagnoses, procedures, age, gender and preauthorization)**
- Generate clear, contextually accurate coverage decisions and provide reasoning for approvals or further routing for manual reviews

Capstone 2 Community

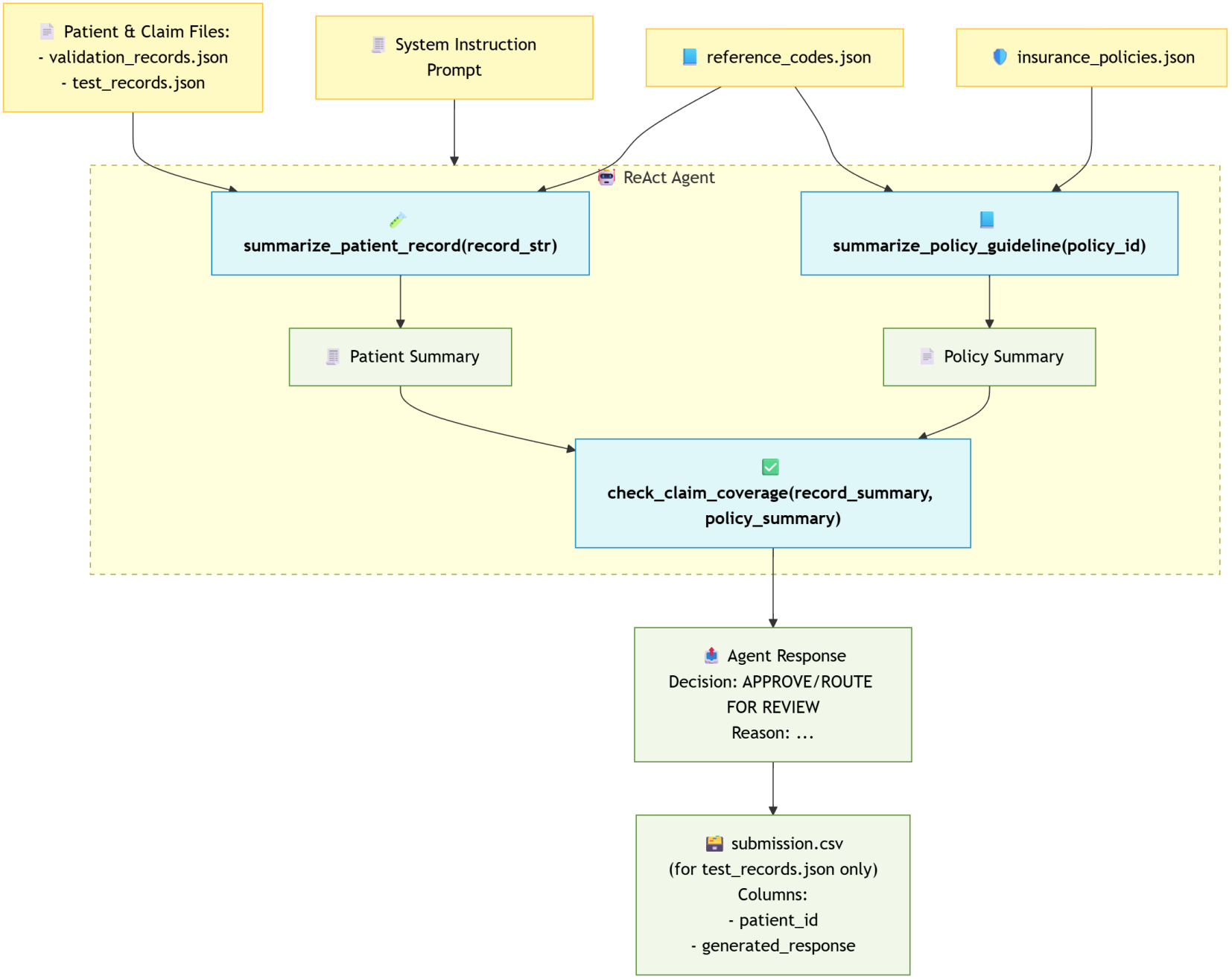
If you would like to connect with peers who are also working through this capstone to ask questions or seek help with troubleshooting, you can join the Capstone 2 Teams channel.

Solution Requirements & Specifications

The Agent Architecture Workflow

The workflow in the following figure shows how your ReAct Agent should function by leveraging a **single agent, three tools,** and **a system instruction prompt**.

- The agent reads patient records and policy documents, then uses **summarize_patient_record** and **summarize_policy_guideline** tools to generate structured summaries.
- It then runs **check_claim_coverage** tool to evaluate whether the claimed procedure meets the coverage criteria.
- The agent produces a final **decision** - either **APPROVE** or **ROUTE FOR REVIEW,** along with a concise **reason**.
- You can use the records from **validation_records.json** to develop, evaluate, and refine your agentic system.
- Once finalized, run your agentic system on the records from **test_records.json** to generate the final outputs, which must be saved in **submission.csv**.


Patient & Claim Files:
- validation_records.json
- test_records.json
System Instruction Prompt
reference_codes.json
insurance_policies.json
ReAct Agent
summarize_patient_record(record_str)
summarize_policy_guideline(policy_id)
Patient Summary
Policy Summary
check_claim_coverage(record_summary, policy_summary)
Agent Response
Decision: APPROVE/ROUTE FOR REVIEW
Reason: ...
submission.csv
(for test_records.json only)
Columns:
- patient_id
- generated_response

The Core System Architecture

The Healthcare Insurance Claim Approval System should be implemented as a Tool-Use ReAct Agent **(Single agent with three tools)** with these key components.

Requirement	Specifications	Notes
Age Computation	Age is one of the most important eligibility criteria in healthcare insurance claims. Policies frequently specify an allowable age range for specific procedures, and claims that fall outside this range must be routed for manual review. To ensure accuracy, the patient's age should be calculated using the following fields from the patient record: `date_of_birth` (e.g., `"1982-03-15"`) `date_of_service` (e.g., `"2025-05-03"`) The age is computed by subtracting the birth year from the service year, and adjusting the result if the patient has not yet had their birthday in the service year. This calculation ensures the age reflects the last fully completed year, which is standard practice in healthcare. For example: If the date of birth is March 15, 1982 and the date of service is May 3, 2025, the computed age is 43. You can compute the age in a preprocessing step using Python and add it directly to each patient record as a key-value pair with "**age**" as the key. For example: **`"age": 43`**	Alternatively, you may choose to compute the age within one of the agent tools, such as *`summarize_patient_record`*, using LLM reasoning. However, it is important to note that large language models have limitations when it comes to performing precise mathematical calculations. Inaccurate age computation can lead to incorrect claim outcomes. The computed age will be used later in the agent's reasoning process to determine whether the patient's age satisfies the policy's claim coverage criteria.

The Tools for the Agent (3 tools)		
Requirement	Specifications	Notes
Summarizing Patient Health Record with Insurance Claim	***summarize_patient_record(record_str)*** • This tool is responsible for extracting a structured summary of a patient's insurance claim record using LLM reasoning. It accepts a **raw patient record** (as a JSON string or plain string) and returns a **well-structured summary report** that will later be used for claim coverage evaluation.	**Key Expectations:** The summary generated by this tool should be clearly formatted and include the following **seven labeled sections**, in the specified order: • **Patient Demographics:** Include: name, gender, and age *(Note: age can be precomputed using Python or LLM reasoning and included in the input record as "age")* • **Insurance Policy ID** • **Diagnoses and Descriptions:** Include ICD-10 codes and their mapped descriptions. • **Procedures and Descriptions:** Include CPT codes and their mapped descriptions. • **Preauthorization Status:** Clearly mention if preauthorization was required and whether it was obtained. • **Billed Amount (in USD)** • **Date of Service**
Additional Notes: • You should use the provided ICD-10 and CPT code mappings (from `reference_codes.json`) to enrich the output with human-readable descriptions of medical codes. These mappings can be passed into the tool and referenced through LLM reasoning to generate clearer, more informative summaries.		

- The summary text should follow a bullet-point format or clearly separated labeled sections, making it easy for downstream tools (and the agent) to reason over the content.
- Avoid adding any decision-making logic here. This tool is focused only on creating a structured summary report, not determining claim approval.

This summary text acts as a standardized input for the claim coverage validation tool that follows.

Summarizing Insurance Policy Details	***summarize_policy_guideline(policy_id)***
	• This tool is responsible for generating a structured summary of the insurance policy corresponding to the given `policy_id`.
	• It retrieves the matching policy document from the dataset and returns a well-formatted summary that outlines the specific claim coverage rules for each procedure under that policy. This summary will later be used to determine whether a patient's claim satisfies the policy's coverage conditions.
		Key Expectations: The summary generated by this tool should be clearly formatted and include the following clearly labeled sections, in order:
		• **Policy Details: Include:** policy ID and plan name
		• **Covered Procedures:** For each covered procedure listed in the policy, include the following sub-points:
		  ○ Procedure Code and Description (using CPT code mappings)
		  ○ Covered Diagnoses and Descriptions (using ICD-10 code mappings)
		  ○ Gender Restriction
		  ○ Age Range
		  ○ Preauthorization Requirement
		  ○ Notes on Coverage (if any)
		
		Each procedure should be presented as a separate entry under the "Covered Procedures" section, with the required sub-points clearly listed.

Additional Notes:

- You should use the provided ICD-10 and CPT code mappings (from `reference_codes.json`) to enrich the output with human-readable descriptions of medical and procedure codes. These mappings can be passed into the tool and referenced through LLM reasoning to produce a more informative summary.
- The summary should follow a consistent bullet-point format or clearly separated labeled sections, which ensure that the agent and downstream tools can interpret the policy data accurately.
- Avoid adding any approval or decision-making logic here. This tool is focused only on presenting the coverage rules as stated in the policy document.

This summary text acts as a standardized input for the claim coverage validation tool that follows.

Validate Insurance Claim Coverage	***check_claim_coverage(record_summary, policy_summary)*** • This tool determines whether the procedures claimed by a patient are covered under their insurance policy. It takes as input the structured summary of the patient record along with the corresponding policy summary generated by the earlier tools. • The tool uses LLM-based reasoning to evaluate each claimed procedure against the applicable policy conditions and returns a coverage eligibility decision, either approval or routing for manual review by a human specialist.	**Coverage Evaluation Criteria:** A procedure should be **approved only if all the following conditions are met:** • The patient's diagnosis code(s) match the policy-covered diagnoses for the claimed procedure. • The procedure code is explicitly listed in the policy, and all associated conditions are satisfied. • The patient's age falls within the policy's defined age range (inclusive of the lower bound, exclusive of the upper bound). • The patient's gender matches the policy's requirement for that procedure. • If preauthorization is required by the policy, it must have been obtained. Only procedures and diagnoses explicitly listed in the patient record should be evaluated. **For simplicity we have kept one procedure per patient**.

Expected Output Format: The tool's response should include the following three sections:

- **Coverage Review:** Step-by-step analysis for the claimed procedure, detailing the checks performed. (each patient has only one procedure)
- **Summary of Findings:** Summary of which coverage requirements were met or not met.
- **Final Decision:** For each procedure for the claim, return either "`APPROVE`" or "`ROUTE FOR REVIEW`" with a brief explanation of the reasoning behind it.

The output of this tool will be used by the agent, through its system instruction prompt, to generate the final decision.

Agent System Instruction Prompt	

Agent System Instruction Prompt	• The system instruction prompt defines the agent's overall reasoning workflow and enforces the use of specific tools in a structured sequence.
	• At a minimum, the prompt should clearly list the three tools the agent is allowed to use
	• Specify the exact sequence in which these tools must be used
	• Define the required final output format, which should have the following:
	  ○ A single-line **Decision**: either **`APPROVE`** or **`ROUTE FOR REVIEW`**
	  ○ A concise **Reason**: that refers to specific coverage rules and policy conditions which led to the above decision by the agent
	You are expected to frame the prompt to enforce this logic. The quality and clarity of the system prompt directly affect the correctness and consistency of the agent's outputs.

The ReAct Agent	

The ReAct Agent	• You are expected to build a ReAct-style **single agent** that uses exactly **three tools** to process insurance claim approvals.
	• You can implement the agent in one of two ways:
	  ○ **Option 1**: Build the agent from scratch using LangGraph, ensuring it is configured as a **single agent with three tools and system prompt**.
	  ○ **Option 2**: Use the built-in `create_react_agent` function from LangGraph, ensuring it is configured as a **single agent with three tools and system prompt**.
	In both cases, the agent should follow the defined tool usage sequence and return decisions in a consistent format.

Technical Requirements

Learners are recommended to leverage the following tools and models to ensure consistency in responses to the extent possible:

Requirement	Description
Large Language Models (LLM)	OpenAI LLM API (GPT-4o-mini)
Development Framework	Orchestration frameworks includes LangChain and LangGraph for ReAct agent development

Dataset Details

To build and test your Insurance Claim Approval Agent, the following datasets are provided as standard JSON & CSV files:

Data File Name	Description/Details	Additional Information

insurance_policies.json	○ This file contains the insurance policy details, including coverage rules, diagnosis-procedure mappings, age ranges, gender eligibility, preauthorization requirements, and additional notes as depicted in a sample policy record in the adjoining column.
	○ This data is used in the following tools:
	○ *summarize_policy_guideline*(policy_id): Retrieves the policy associated with a given ID and generates a structured summary of its contents. This includes the procedures covered and their associated requirements, enriched with CPT and ICD-10 descriptions.
		<pre>{
		'policy_id': 'POL1007',
		'plan_name': 'Essential Care Plus',
		'covered_procedures': [
		{
		'procedure_code': '85025',
		'covered_diagnoses': ['N39.0'],
		'age_range': [28, 79],
		'gender': 'Any',
		'requires_preauthorization': True,
		'notes': 'Complete blood count (CBC) covered for N39.0 cases.'},
		{
		'procedure_code': '93010',
		'covered_diagnoses': ['N39.0'],
		'age_range': [1, 70],
		'gender': 'Male',
		'requires_preauthorization': False,
		'notes': 'ECG interpretation only covered for N39.0 cases.'
		}
		]
		}</pre>

reference_codes.json	○ This dictionary provides human-readable descriptions for:	<pre>{
		'93000': 'Electrocardiogram, routine ECG with at least 12 leads; with interpretation and report',</pre>

	○ CPT (procedure) codes
	○ ICD-10 (diagnosis) codes
	○ Use these mappings to enrich the output of your patient and policy summary tools. The mappings can be passed to the tools and referenced using LLM reasoning to improve clarity and explainability.
		<pre>'99213': 'Office or outpatient visit for an established patient, typically 20-29 minutes',
		...,
		...
		}</pre>

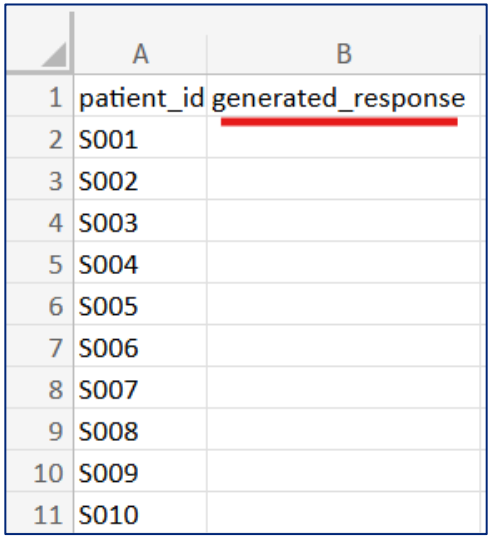
validation_records.json	○ This file contains 10 sample patient claim records.
	○ Each record includes the patient's demographics, diagnosis codes, procedure codes, preauthorization status, billed amount, and other relevant metadata. For simplicity, the patient and claim data have been combined into a single record per patient as depicted in a sample patient record as shown in the adjoining column.
	○ These records should be used during development to experiment and evaluate and make sure that your agent is functioning correctly, before proceeding to the test dataset.
	○ You are expected to compute the patient's age using the `date_of_birth` and `date_of_service` fields and add it as an "age" key in each record to ensure correct agent reasoning.
	○ The corresponding human reference responses are available in **`validation_reference_results.csv`**
		<pre>{
		'patient_id': 'P002',
		'name': 'Robert Jones',
		'date_of_birth': '1982-03-15',
		'gender': 'Female',
		'insurance_policy_id': 'POL1007',
		'diagnosis_codes': ['N39.0'],
		'procedure_codes': ['85025'],
		'date_of_service': '2025-05-03',
		'provider_id': 'PRV002',
		'provider_specialty': 'Neurology',
		'location': 'Los Angeles, CA',
		'billed_amount': 3500.0,
		'preauthorization_required': True,
		'preauthorization_obtained': True,
		'age': 43 # you will need to compute this field in your code
		}</pre>

<table>
<tr>
<td></td>
<td>

These examples are provided to help you assess your agent's reasoning and output format by comparing its responses to the human reference responses.
This can be done through manual inspection or by designing simple LLM as a Judge comparison checks with basic prompt engineering.

</td>
<td></td>
</tr>
<tr>
<td><code>test_records.json</code></td>
<td>

This file contains 10 unseen patient claim records that will be used for final evaluation.
Be sure to include the <code>age</code> field, formatted the same way as in the evaluation records.
You must run these records through your agent and record the final response for each one.
Results should be stored in a file named <code>submission.csv</code> and submitted as part of your deliverables along with the code notebook.
The result file should contain two columns: <code>patient_id</code> and <code>generated_response</code>, corresponding to each patient in the test dataset.
The <code>generated_response</code> column must be populated with the agent's output for that specific patient ID. A blank sample file named

</td>
<td>

<table>
<tr><th></th><th>A</th><th>B</th></tr>
<tr><td>1</td><td>patient_id</td><td>generated_response</td></tr>
<tr><td>2</td><td>S001</td><td></td></tr>
<tr><td>3</td><td>S002</td><td></td></tr>
<tr><td>4</td><td>S003</td><td></td></tr>
<tr><td>5</td><td>S004</td><td></td></tr>
<tr><td>6</td><td>S005</td><td></td></tr>
<tr><td>7</td><td>S006</td><td></td></tr>
<tr><td>8</td><td>S007</td><td></td></tr>
<tr><td>9</td><td>S008</td><td></td></tr>
<tr><td>10</td><td>S009</td><td></td></tr>
<tr><td>11</td><td>S010</td><td></td></tr>
</table>
</td>
</tr>
</table>

	`submission.csv` is provided for reference, following this exact format, following this link.	

Building Your Healthcare Insurance Claim Approval Agent

Create a notebook named "**`code.ipynb`**" where you will build your Claim Approval Agent. Your Jupyter Notebook (`code.ipynb`) must include the complete implementation of your Healthcare Insurance Claim Approval Agent architecture. This notebook will serve as the primary artifact for grading. Your final submission should include the following:

- Jupyter Notebook containing the Insurance Claim Approval Agent Implementation code (code file in ipynb format)
- Based on the Test Dataset Provided (`test_records.json`), run it on your agentic system and submit the following (as mentioned earlier):
    - CSV file (`submission.csv`) containing the following:
        - **Patient ID** (column: `patient_id`)
        - **Agent Final Response** (Column: `generated_response`)

Submission Guidelines:

Your Jupyter Notebook (`code.ipynb`) should contain the complete implementation of your **Healthcare Insurance Claim Approval Agent** and must include the following:

- The notebook should be clearly structured using appropriate **Markdown headings** and **code comments** to explain each section.
- Each part of the implementation should be easy to follow and logically organized, demonstrating how it fits into the overall agent system.
- Required Components in the Notebook:

S No	Component	Description

1	**Dataset Loading**	• Load all required input files (standard JSON files):
		  ○ **`reference_codes.json`** (ICD-10 and CPT code mappings)
		  ○ **`validation_records.json`** & **`test_records.json`** (patient records)
		  ○ **`insurance_policies.json`** (policy documents)

2	**Tool Definitions & Implementations**	• Define & implement the following tools:
		  ○ **`summarize_patient_record(record_str)`**
		  ○ **`summarize_policy_guideline(policy_id)`**
		  ○ **`check_claim_coverage(record_summary, policy_summary)`**
		• Ensure that each tool adheres to the specifications outlined in the project description, including output format and structure.

3	**System Instruction Prompt**	• Implement a system instruction prompt that:
		  ○ Clearly defines the three tools available to the agent
		  ○ Specifies the workflow sequence the agent must follow
		  ○ Enforces the correct output format (Decision and Reason)

4	**LLM Integration**	• Use the selected LLM (e.g., gpt-4o-mini) using a LangChain-compatible setup.
5	**Agent Creation**	Create a **ReAct-style single agent using LangGraph**: • **Option 1: Manually construct** the agent with defined logic • **Option 2:** Use the built-in **`create_react_agent()`** method with proper configuration
6	**Agent Experimentation & Validation**	• For each patient record in **`validation_records.json`**, send the input to the agent • Note and analyze the agent's response, and ensure that everything is functioning as expected.

		• Load the human reference responses from the **`validation_reference_results.csv`** file • Evaluate your agent using manual inspection or by applying simple "LLM as a Judge" prompting to compare the agent's response with the human reference response. • Adjust and improve your agent accordingly
7	**Testing**	Generate agent responses for the patients in **`test_records.json`** and store them in the format described in the following section.

Final Output

Your final output file must be named **`submission.csv`** and should contain the generated agent responses for the test dataset (**`test_records.json`**). This file will be used to evaluate your agent's performance.

- A sample file with blank responses is provided to you (**`submission.csv`**) for reference as depicted in the following snapshot. You are expected to fill in the **`generated_response`** column for each patient in the test data.

	A	B
1	patient_id	generated_response
2	S001	
3	S002	
4	S003	
5	S004	
6	S005	
7	S006	
8	S007	
9	S008	
10	S009	
11	S010	

- **Format Requirements:** The final CSV file (**`submission.csv`**) must have the following **two columns**, in the exact order and should look something like the following snapshot

patient_id	generated_response
S001	- **Decision:** ROUTE FOR REVIEW - **Reason:** The claim for CPT code 36415 cannot be automatically approved due to policy restrictions and must be routed for further manual review.
S002	- **Decision:** APPROVE - **Reason:** The claim for CPT code 85025 is approved...
S003	...
S004	...

  - **Column 1: `patient_id`**
    The unique ID of the patient from **test_records.json** (e.g., S001, S002, etc.).
  - **Column 2: `generated_response`**
    The final output from your agent for that patient record. The format must follow the structure defined in your agent's system instruction prompt and it should follow the same format and style as the sample responses shown in the snapshot above.

The **`submission.csv`** file and the **`code.ipynb`** files will be the **primary artifacts used for final evaluation**, so make sure it is complete, consistent, and correctly formatted. *We recommend you review the submission checklist on page 30 before submitting your files.*

How Your Submission will be Evaluated

The submission would be evaluated by looking at several aspects, including:

- Performance metrics measuring final response accuracy and reasoning quality
- Code Quality
- ReAct Agentic workflow used in the code

The following 3-point Rubrics (with a maximum scoring opportunity of 12), would be used for scoring your submission:

Component	Definition	Scoring Guidelines	Needs Improvement (1 point)	Satisfactory (2 points)	Excellent (3 points)
Agentic AI Performance - Response Reasoning Quality	Assesses the quality of the agent's final response and underlying reasoning across all 10 test cases, based on patient records and their corresponding policy details	Evaluation will be based on two core performance metrics: **Response Relevance** and **Hallucination Score**	Responses demonstrate incorrect or unclear reasoning. CPT/ICD codes are misused, misinterpreted, or hallucinated. References to patient or policy data are missing, irrelevant, or factually incorrect. Frequent hallucinations, including fabricated procedures, diagnoses, or policy terms.	The majority (>50%, but <80%) of responses show partially correct or contextually relevant reasoning. Correct CPT/ICD codes are used in most cases, with basic alignment to patient diagnoses and policy coverage. Minor hallucinations may be present but do not significantly affect the final decision. Final outcomes mostly	Reasoning is accurate, complete, and well-aligned in almost all responses (>=80%). Correct CPT/ICD codes are consistently applied, explicitly linked to patient conditions and policy eligibility. No hallucinations—agent never fabricates procedures, diagnoses, or policy rules. Final decisions are clearly structured, logically justified, and supported by explicit references to both patient and policy data.

				align with reference responses, though reasoning may lack depth or completeness.	
	Definition	**Scoring Guidelines**	**Needs Improvement (1 point)**	**Satisfactory (2 points)**	**Excellent (3 points)**
Agentic AI Performance – Decision Accuracy	Measures how accurately the agent classifies insurance claims as **APPROVE** or **ROUTE TO REVIEW** across all 10 test cases, based on comparison with ground-truth reference responses.	**Evaluation** will leverage standard classification metrics like **Accuracy**, **Precision**, **Recall**, and **F1-score** across the 10 test cases and decide the final score based on these metrics.	Most responses exhibit low performance across key metrics (precision, recall, and F1-score) particularly for one or both classes. The agent demonstrates biased or random behavior (e.g., always selecting "APPROVE"), resulting in poor alignment with reference decisions.	The majority (>50%, but <80%) of responses are correctly classified, with moderate precision and recall scores. The agent handles common scenarios reasonably well, though struggles with edge cases or complex policy conditions.	Almost all responses (>=80%) are correctly classified, with consistently high precision, recall, and F1-score across both APPROVE and ROUTE TO REVIEW classes. Final outcomes closely match the reference decisions, with strong alignment across all test cases.
	Definition	**Scoring Guidelines**	**Needs Improvement (1 point)**	**Satisfactory (2 points)**	**Excellent (3 points)**
Solution Code Quality	Evaluates the overall quality of the code implementing the solution notebook, focusing on readability,		Code is difficult to read or understand, lacks proper structure, and contains minimal or no documentation	Code is mostly modular, with some level of documentation and comments. Structure and	Code is clean, well-structured, and modular. It includes comprehensive documentation, comments, clear variable and function naming, and adheres to

	modularity, correctness, and documentation.		or comments. Error handling is inadequate or absent.	readability are acceptable, though there may be minor issues.	best practices for readability, modularity, and error handling.
	Definition	**Scoring Guidelines**	**Needs Improvement (1 point)**	**Satisfactory (2 points)**	**Excellent (3 points)**
Agentic Workflow Design & Implementation	Evaluates the overall architecture and implementation of the agentic system pipeline, including tool design, configuration, planning logic, step-wise execution, and response generation across a **multi-tool, single ReAct Agent** workflow.	Evaluation will be based on the system's completeness, logical flow, tool integration, prompt quality, and overall workflow.	The single ReAct agentic system is incomplete or contains major logical flaws. Tools are used incorrectly or misaligned (e.g., applying the policy summarizer to patient data). Workflow steps are missing or mentioned in the wrong order (e.g., performing eligibility checks before summarization). The quality of individual tool-level prompts and system-level prompts is poor and overly generic. ReAct agent	A standard end-to-end single ReAct agentic system is implemented with a basic correct flow: **Summarize Record → Summarize Policy → Validate Eligibility** Utilizes LangChain (LLM I/O) and standard LangGraph flows. All tools are appropriately configured and integrated with proper input-output handling. Includes: • Basic tool definitions and prompts • Defined ReAct agent	A robust, modular, and well-orchestrated end-to-end single ReAct agentic system is implemented with a proper expected flow: **Summarize Record → Summarize Policy → Validate Eligibility → Final Decision with Detailed Reasoning** Utilizes LangChain (LLM I/O) and standard LangGraph flows. All tools are appropriately configured and integrated with proper input-output handling. Includes: • Well-crafted system-level prompts that clearly define the agent's behavior and decision-making objectives • Detailed and specific tool-level prompts to guide each tool's function effectively • A clearly defined output format for structured and consistent responses

			workflow code has flaws.	• Clear tool orchestration • May lack clarity in tool-level or system-level prompt instructions and output response formatting instructions and consistency	• Final responses include justifications and reasoning for decisions (approval/rejection), backed by patient and policy data • Use of prompt engineering techniques to minimize ambiguity, enhance response quality, and ensure tool alignment

Total Score	Grade	Pass/Fail	Description
11–12 points	**Excellent**	**Pass**	Outstanding performance across all areas. Demonstrates a strong understanding of RAG concepts and implementation.
8–10 points	**Satisfactory**	**Pass**	Good overall performance with a solid grasp of key concepts, though there is room for improvement in some areas.
4–7 points	**Needs Improvement**	**Fail**	Basic or inconsistent performance. Key areas of the RAG system require further development and refinement.

Note to Learners

About Evaluation

Once you submit your capstone, we will evaluate it within 14 days. Once your capstone evaluation is complete, you will receive a detailed report regarding your submission. If you received a passing grade, this will be visible in the Generative AI journey in MyLearning. If you do not receive your grade within 2 weeks of your submission, you can send us an email or drop in a comment in the AIML community.

About Resubmission

Your first submission of the Generative AI capstone is free, and you get one free resubmission if you don't pass the first time. If you fail the capstone twice, your third (and every subsequent) submission will incur a $100 charge to your department's GL. This is to encourage you to do your very best work the first time you submit!

Submission Checklist

Sl No.	Description	Status
1	The Jupyter notebook is saved as **`code.ipynb`**	☐ Checked
2	The Jupyter notebook has all the major sections including: • Markdown headings and code comments • Logical and organized implementation of the overall agent system	☐ Checked
3	Dataset loading – with all necessary input files	☐ Checked
4	Tool definitions and implementations with proper definition and implementation of the following tools: • **`summarize_patient_record(record_str)`** • **`summarize_policy_guideline(policy_id)`**	☐ Checked

	• **`check_claim_coverage(record_summary, policy_summary)`**	
5	Each tool adheres to the specifications outlined in the project description, including output format and structure.	☐ Checked
6	The system instruction prompts: • Clearly defines the three tools available to the agent • Specifies the workflow sequence the agent must follow • Enforces the correct output format (Decision and Reason)	☐ Checked
7	LLM selected using a LangChain compatible setup.	☐ Checked
8	The Single Agent created using LangGraph is a ReAct agent.	☐ Checked
9	The agent's responses for all records in **`validation_records.json`** were compared with the human reference answers in **`validation_reference_results.csv`** to verify that the system is functioning as expected.	☐ Checked
10	The final output for test results (after generating the agent's responses for all records in **`test_records.json)`** is named as "**`submission.csv`**" and it follows the recommended CSV structure with column names "`patient_id`" and "`generated_response`".	☐ Checked

Appendix

Sample Data Example:

CPT Code Mapping Dataset (used to enrich summary reports)

```
{
  '93000': 'Electrocardiogram, routine ECG with at least 12 leads; with interpretation and report',
  '99213': 'Office or outpatient visit for an established patient, typically 20–29 minutes',
  '70450': 'CT scan of the head or brain without contrast material',
  '83036': 'Hemoglobin glycosylated A1c test',
  '93010': 'ECG interpretation and reporting of a routine ECG with at least 12 leads',
  '99214': 'Office visit for established patient with moderate complexity',
  '71200': 'Chest X-ray, two views, frontal and lateral',
  '36415': 'Collection of venous blood by venipuncture',
  '85025': 'Complete blood count (CBC) with automated differential',
  '45378': 'Diagnostic colonoscopy including collection of specimens'
}
```

ICD-10 Code Mapping Dataset (used to enrich summary reports):

```
{
  'I10': 'Essential (primary) hypertension',
```

```
  'E11.9': 'Type 2 diabetes mellitus without complications',
  'G43.909': 'Migraine, unspecified, not intractable, without status migrainosus',
  'I20.0': 'Unstable angina',
  'I48.91': 'Unspecified atrial fibrillation',
  'J44.9': 'Chronic obstructive pulmonary disease, unspecified',
  'K21.9': 'Gastro-esophageal reflux disease without esophagitis',
  'M54.5': 'Low back pain, also known as lumbago',
  'N39.0': 'Urinary tract infection, site of infection not specified',
  'F32.9': 'Major depressive disorder, single episode, unspecified'
}
```

Patient Detail Example (Input to the Agent):

```
{
'patient_id': 'P002',
      'name': 'Robert Jones',
      'date_of_birth': '1982-03-15',
      'gender': 'Female',
      'insurance_policy_id': 'POL1007',
      'diagnosis_codes': ['N39.0'],
      'procedure_codes': ['85025'],
      'date_of_service': '2025-05-03',
      'provider_id': 'PRV002',
      'provider_specialty': 'Neurology',
      'location': 'Los Angeles, CA',
      'claim_amount': 3500.0,
```

```
        'preauthorization_required': True,
        'preauthorization_obtained': True,
        'age': 43  # needs to be computed

    }
```

Policy Document Guidelines Example (Used by the Agent):

```
    {
    'policy_id': 'POL1007',
        'plan_name': 'Essential Care Plus',
        'covered_procedures': [
            {
            'procedure_code': '85025',
                'covered_diagnoses': ['N39.0'],
                'age_range': [28, 79],
                'gender': 'Any',
                'requires_preauthorization': True,
                'coverage_limit': 5947.37,
                'notes': 'Complete blood count (CBC) covered for N39.0 cases.'
            },
            {
            'procedure_code': '93010',
                'covered_diagnoses': ['N39.0'],
                'age_range': [1, 70],
                'gender': 'Male',
                'requires_preauthorization': False,
```

```
                'coverage_limit': 3947.91,
                'notes': 'ECG interpretation only covered for N39.0 cases.'
            }
    ]}
```

Sample Agent Response:

- **Decision:** APPROVE
- **Reason:** The claim for the complete blood count (CPT code 85025) is approved because this procedure is covered under the policy for the diagnosis of urinary tract infection (N39.0), which applies to the patient's diagnosis. The patient's age (43) and gender (Female) meet the policy requirements, and preauthorization was obtained. Additionally, the claim amount of $3500.00 would be paid in full, as the bill amount is less than the permissible amount of $5947.37. All coverage requirements are satisfied.

Sample Agent Response (for manual review routing):

- **Decision:** ROUTE FOR REVIEW
- **Reason:** The claim for the electrocardiogram (CPT code 93000) cannot be automatically approved because the patient's age of 69 exceeds the policy's allowed age range of 11 to 63. Although the diagnosis of low back pain (M54.5) and the procedure code match the policy's covered conditions, the age requirement is not met. Therefore, the claim needs to be routed for further manual review.