Technical Report and Appendix

Models Used

The models used in this project are listed below:
Proprietary Advanced: GPT-5
Proprietary Standard:
Open Source: Gemma3:4b

Setup and Configuration

- The models were used/accessed directly from their respective chat GUIs.
- For Gemma3, the model was downloaded to the local system by using Ollama, and Enchanted was used to access the model via its GUI.
- No Setup challenges were encountered during the installation or configuration of any model.

Technical Limitations

- No technical limitations were found or encountered.
- All three models were able to process the inputs smoothly and generate outputs consistently.

Tokens Usage

- The inputs for all three models were the same throughout, and the outputs were quite similar as well. Thus, the token usage should be the same, too. The only difference would come when "Thinking and Reasoning" models were being used, as the token usage would have increased significantly at that time.
- Chain-of-Thought prompts must have increased the token usage due to step-by-step reasoning.

Response Time

- The model response time varied.
- Gemma generally gave the results the fastest.
- GPT-5 and Gemini Pro triggered their Reasoning and Thinking models, which took more time to generate the output, but generally both models were generating responses at the same time.

PS: While GPT-5 and Gemini Pro were slower due to complex reasoning, their latency remained acceptable.

All the actual prompts used are listed below:

Task 1: Code Summarization or Information Synthesis

CLEAR

You are a senior software engineer tasked with reviewing a code snippet and providing a clear, concise summary for other developers. Context: You will be given: 1. The code snippet 2. A little description about the project Instructions: - Summarize the code's purpose in 2–3 sentences. - List the key functions or classes and their roles. - Describe the inputs and outputs clearly. - Mention any dependencies or external libraries used. - Use simple, professional language appropriate for internal documentation. - Limit the response to 500 words Code: import jakarta.servlet.ServletException; import jakarta.servlet.annotation.WebServlet; import jakarta.servlet.http.HttpServlet; import jakarta.servlet.http.HttpServletRequest; import jakarta.servlet.http.HttpServletResponse; import jakarta.xml.bind.DatatypeConverter; import java.io.IOException; import java.io.PrintWriter; import java.security.MessageDigest; import java.security.NoSuchAlgorithmException; @WebServlet(name = "ComputeHashes", value = {"/computeHashes"}) public class ComputeHashes extends HttpServlet { protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException { String text = request.getParameter("text"); String algo = request.getParameter("algo"); if (text == null) { text = ""; } if (algo == null || !algo.equalsIgnoreCase("MD5") && !algo.equalsIgnoreCase("SHA-256")) { algo = "MD5"; } byte[] hashBytes; try { MessageDigest md = MessageDigest.getInstance(algo); hashBytes = md.digest(text.getBytes()); } catch (NoSuchAlgorithmException var9) { response.getWriter().println("Error: Unsupported algorithm"); return; } String hex = DatatypeConverter.printHexBinary(hashBytes); String base64 = DatatypeConverter.printBase64Binary(hashBytes); response.setContentType("text/html"); PrintWriter out = response.getWriter(); out.println("<html>"); out.println("<head><title>Compute Hash</title></head>"); out.println("<body>"); out.println("<h2>Hash Result</h2>"); out.println(" Algorithm: " + algo + " "); out.println(" Original Text: " + text + " "); out.println(" Hex: " + hex + " "); out.println(" Base64: " + base64 + " "); out.println(" Back "); out.println("</body></html>"); } } Project Description: Create an index.jsp page that asks the user to enter a string of text data, and to make a choice of two hash functions using radio buttons. The hash function choices should be MD5 and SHA-256, with MD5 being the default. When the submit button is pressed a request is sent to a servlet. The servlet must be named ComputeHashes.java. The servlet will compute the requested cryptographic hash value (MD5 or SHA-256) from the text transmitted by the browser. You will need to employ the Java crypto API to compute the hash of the text. The original text will be echoed back to the browser along with the name of the hash, and the hash value. The hash values sent back to the browser should be displayed in two forms: as hexadecimal text and as base 64 notation.

Few-Shot

You are tasked with summarizing code snippets for documentation. Here are two examples: Example 1: Code: def add(a, b): return a + b Summary: This function add takes two numbers and returns their sum. Inputs: a, b (numbers). Output: number (sum). Example 2: Code: class Order: def __init__(self, id, amount): self.id = id self.amount = amount Summary: The Order class represents a purchase order, including its ID and amount. Constructor initializes these attributes. Inputs: id, amount. Output: instantiated Order object. Now, summarize the following code using the same style: Code: import

jakarta.servlet.ServletException; import jakarta.servlet.annotation.WebServlet; import jakarta.servlet.http.HttpServlet; import jakarta.servlet.http.HttpServletRequest; import jakarta.servlet.http.HttpServletResponse; import jakarta.xml.bind.DatatypeConverter; import java.io.IOException; import java.io.PrintWriter; import java.security.MessageDigest; import java.security.NoSuchAlgorithmException; @WebServlet(name = "ComputeHashes", value = {"/computeHashes"}) public class ComputeHashes extends HttpServlet { protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException { String text = request.getParameter("text"); String algo = request.getParameter("algo"); if (text == null) { text = ""; } if (algo == null || !algo.equalsIgnoreCase("MD5") && !algo.equalsIgnoreCase("SHA-256")) { algo = "MD5"; } byte[] hashBytes; try { MessageDigest md = MessageDigest.getInstance(algo); hashBytes = md.digest(text.getBytes()); } catch (NoSuchAlgorithmException var9) { response.getWriter().println("Error: Unsupported algorithm"); return; } String hex = DatatypeConverter.printHexBinary(hashBytes); String base64 = DatatypeConverter.printBase64Binary(hashBytes); response.setContentType("text/html"); PrintWriter out = response.getWriter(); out.println("<html>"); out.println("<head><title>Compute Hash</title></head>"); out.println("<body>"); out.println("<h2>Hash Result</h2>"); out.println(" Algorithm: " + algo + " "); out.println(" Original Text: " + text + " "); out.println(" Hex: " + hex + " "); out.println(" Base64: " + base64 + " "); out.println(" Back "); out.println("</body></html>"); } }

Chain-of-Thought

You are analyzing a code snippet to create a summary for documentation. First, follow these reasoning steps:1. Identify the main purpose of the code.2. List each function or class and briefly describe what it does.3. Determine the inputs and outputs for the code.4. Check for external libraries or dependencies.5. Combine this information into a concise 2–3 sentence summary.Code:import jakarta.servlet.ServletException;import jakarta.servlet.annotation.WebServlet;import jakarta.servlet.http.HttpServlet;import jakarta.servlet.http.HttpServletRequest;import jakarta.servlet.http.HttpServletResponse;import jakarta.xml.bind.DatatypeConverter;import java.io.IOException;import java.io.PrintWriter;import java.security.MessageDigest;import java.security.NoSuchAlgorithmException;@WebServlet(name = "ComputeHashes", value = {"/computeHashes"})public class ComputeHashes extends HttpServlet { protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException { String text = request.getParameter("text"); String algo = request.getParameter("algo"); if (text == null) { text = ""; } if (algo == null || !algo.equalsIgnoreCase("MD5") && !algo.equalsIgnoreCase("SHA-256")) { algo = "MD5"; } byte[] hashBytes; try { MessageDigest md = MessageDigest.getInstance(algo); hashBytes = md.digest(text.getBytes()); } catch (NoSuchAlgorithmException var9) { response.getWriter().println("Error: Unsupported algorithm"); return; } String hex = DatatypeConverter.printHexBinary(hashBytes); String base64 = DatatypeConverter.printBase64Binary(hashBytes); response.setContentType("text/html"); PrintWriter out = response.getWriter(); out.println("<html>"); out.println("<head><title>Compute Hash</title></head>"); out.println("<body>"); out.println("<h2>Hash Result</h2>"); out.println(" Algorithm: " + algo + " "); out.println(" Original Text: " + text + " "); out.println(" Hex: " + hex + " "); out.println(" Base64: " + base64 + " ");

out.println(" Back ");
out.println("</body></html>"); }}Final Summary (concise and professional)

Task 2

CLEAR
You are an experienced API support engineer. Your job is to answer developer questions accurately and clearly. Answer using ONLY the provided documentation.

Context:
You will receive:
1. Developer's question
2. Relevant section of the documentation

Instructions:
- Provide a direct, actionable answer.
- Include a minimal example if helpful.
- Cite the exact section or heading from the documentation.
- Do NOT invent information beyond the provided documentation.
- Limit the response to 500 words

Question:
How do I securely authenticate API requests to Stripe, and what are the risks of exposing secret keys?

Documentation:
The Stripe API uses API keys to authenticate requests. You can view and manage your API keys in the Stripe Dashboard.
Test mode secret keys have the prefix sk_test_ and live mode secret keys have the prefix sk_live_. Alternatively, you can use restricted API keys for granular permissions.
Your API keys carry many privileges, so be sure to keep them secure! Do not share your secret API keys in publicly accessible areas such as GitHub, client-side code, and so forth.
Use your API key by assigning it to stripe.api_key. The Python library will then automatically send this key in each request.
You can also set a per-request key with an option. This is often useful for Connect applications that use multiple API keys during the lifetime of a process. Methods on the returned object reuse the same API key.
All API requests must be made over HTTPS. Calls made over plain HTTP will fail. API requests without authentication will also fail.
Global API Key
Server-side language

Python
import stripe
stripe.api_key =
"sk_test_51S8L3e7K0C8xLGzqCfjLIxwIF3DhBMANhn8OiPDGdds5WaWMsLPJdlVJspZOUOTzFdQqm3vIgughjvTHA5RUq7mV00hEa2fzPM"
Per-Request API Key
Server-side language

Python
import stripe

```
charge = stripe.Charge.retrieve(
  "ch_3Ln3e92eZvKYlo2C0eUfv7bi",

api_key="sk_test_51S8L3e7K0C8xLGzqCfjLIxwIF3DhBMANhn8OiPDGdds5WaWMsLPJdlVJspZ
OUOTzFdQqm3vIgughjvTHA5RUq7mV00hEa2fzPM"
)
charge.capture() # Uses the same API Key.
```

Few-Shot
You are answering developer questions based on documentation.

Example 1:
Question: How do I authenticate with the API?
Docs: "To authenticate, include your API key in the Authorization header."
Answer: Include your API key in the `Authorization` header as follows:
Authorization: Bearer <your_api_key>

Example 2:
Question: How do I create a new user?
Docs: "POST /users creates a new user. Required fields: email, password."
Answer: Use the `POST /users` endpoint with `email` and `password` fields:

```
{
  "email": "test@example.com",
  "password": "mypassword"
}
```

Now, answer the following question using the documentation:
Question:
How do I securely authenticate API requests to Stripe, and what are the risks of exposing secret keys?

Docs:
The Stripe API uses API keys to authenticate requests. You can view and manage your API keys in the Stripe Dashboard.
Test mode secret keys have the prefix sk_test_ and live mode secret keys have the prefix sk_live_. Alternatively, you can use restricted API keys for granular permissions.
Your API keys carry many privileges, so be sure to keep them secure! Do not share your secret API keys in publicly accessible areas such as GitHub, client-side code, and so forth.
Use your API key by assigning it to stripe.api_key. The Python library will then automatically send this key in each request.
You can also set a per-request key with an option. This is often useful for Connect applications that use multiple API keys during the lifetime of a process. Methods on the returned object reuse the same API key.
All API requests must be made over HTTPS. Calls made over plain HTTP will fail. API requests without authentication will also fail.
Global API Key
Server-side language

Python
import stripe
stripe.api_key =
"sk_test_51S8L3e7K0C8xLGzqCfjLIxwIF3DhBMANhn8OiPDGdds5WaWMsLPJdlVJspZOUOTzFdQqm3vIgughjvTHA5RUq7mV00hEa2fzPM"
Per-Request API Key
Server-side language

Python
import stripe
charge = stripe.Charge.retrieve(
  "ch_3Ln3e92eZvKYlo2C0eUfv7bi",

api_key="sk_test_51S8L3e7K0C8xLGzqCfjLIxwIF3DhBMANhn8OiPDGdds5WaWMsLPJdlVJspZOUOTzFdQqm3vIgughjvTHA5RUq7mV00hEa2fzPM"
)
charge.capture() # Uses the same API Key.

Chain-of-thought
You are answering a developer question. Follow these reasoning steps:
1. Read the question carefully.
2. Identify key action or entity being asked about.
3. Scan the provided documentation for relevant information.
4. Match the question to the exact documentation content.
5. Formulate a clear, concise answer with a code example if applicable.

Question:
How do I securely authenticate API requests to Stripe, and what are the risks of exposing secret keys?

Docs:
The Stripe API uses API keys to authenticate requests. You can view and manage your API keys in the Stripe Dashboard.
Test mode secret keys have the prefix sk_test_ and live mode secret keys have the prefix sk_live_. Alternatively, you can use restricted API keys for granular permissions.
Your API keys carry many privileges, so be sure to keep them secure! Do not share your secret API keys in publicly accessible areas such as GitHub, client-side code, and so forth.
Use your API key by assigning it to stripe.api_key. The Python library will then automatically send this key in each request.
You can also set a per-request key with an option. This is often useful for Connect applications that use multiple API keys during the lifetime of a process. Methods on the returned object reuse the same API key.
All API requests must be made over HTTPS. Calls made over plain HTTP will fail. API requests without authentication will also fail.
Global API Key
Server-side language

Python
import stripe
stripe.api_key =
"sk_test_51S8L3e7K0C8xLGzqCfjLIxwIF3DhBMANhn8OiPDGdds5WaWMsLPJdlVJspZOUOTzF
dQqm3vIgughjvTHA5RUq7mV00hEa2fzPM"
Per-Request API Key
Server-side language

Python
import stripe
charge = stripe.Charge.retrieve(
  "ch_3Ln3e92eZvKYlo2C0eUfv7bi",

api_key="sk_test_51S8L3e7K0C8xLGzqCfjLIxwIF3DhBMANhn8OiPDGdds5WaWMsLPJdlVJspZ
OUOTzFdQqm3vIgughjvTHA5RUq7mV00hEa2fzPM"
)
charge.capture() # Uses the same API Key.

Task 3

CLEAR
You are a technical writer and DevRel expert creating professional API documentation.

Instructions:
Use ONLY the given API specification.
Your output must include these sections in this exact order:
1. Title
2. Description
3. Endpoint & Method
4. Headers
5. Parameters (Markdown Table: Name | Type | Required | Description)
6. Example Request
7. Example Response
8. Errors
9. Notes / Best Practices

- If a field or error is not in the spec, write: "Not specified in the spec."
- Do NOT invent information.
- Limit response to 400-500 words.

API Spec:
Endpoint: POST /v1/customers
Description: Create a new customer object.
Headers:
- Authorization: Bearer <sk_test_xxx> (required)
- Content-Type: application/json (required)

Request Body (JSON, optional):
- metadata: object (optional) — arbitrary key/value pairs. Example: {"order_id":"6735"}

Responses:
- 200 OK:
  body (example excerpt):
  id: "cus_123456789" (string)
  object: "customer" (string)
  address: object | null
  balance: integer
  created: integer (unix timestamp)
  currency: string | null
  default_source: string | null
  delinquent: boolean
  description: string | null
  discount: object | null
  email: string | null
  invoice_prefix: string
  invoice_settings: object
  livemode: boolean
  metadata: object
  name: string | null

next_invoice_sequence: integer
phone: string | null
preferred_locales: array<string>
shipping: object | null
tax_exempt: "none" | "exempt" | "reverse"
Notes:
- Use HTTPS only.
- Never expose secret keys in client-side code.
- Include the secret key server-side via Authorization header: "Authorization: Bearer <sk_test_xxx>".

Few-Shot
You are writing structured API documentation. Use ONLY the given spec.

Example:
Spec:
Endpoint: POST /orders
Headers: Authorization, Content-Type
Fields: customer_id, items[]
Description: Creates a new order.

Output:
POST /orders
Creates a new order with one or more line items.

Headers
Name	Required	Description
Authorization	Yes	Bearer token for authentication
Content-Type	Yes	Must be application/json

Parameters
Name	Type	Required	Description
customer_id	string	Yes	Unique customer identifier
items	array	Yes	Array of order items

Example Request
POST /orders
```
{
  "customer_id": "12345",
  "items": [
    { "sku": "SKU-001", "quantity": 2 }
  ]
}
```

Errors
- 400 Bad Request – Invalid parameters
- 401 Unauthorized – Missing or invalid token

Now, generate documentation for the following:

API Spec:
Endpoint: POST /v1/customers
Description: Create a new customer object.
Headers:
  - Authorization: Bearer <sk_test_xxx> (required)
  - Content-Type: application/json (required)
Request Body (JSON, optional):
  - metadata: object (optional) — arbitrary key/value pairs. Example: {"order_id":"6735"}
Responses:
  - 200 OK:
      body (example excerpt):
        id: "cus_123456789" (string)
        object: "customer" (string)
        address: object | null
        balance: integer
        created: integer (unix timestamp)
        currency: string | null
        default_source: string | null
        delinquent: boolean
        description: string | null
        discount: object | null
        email: string | null
        invoice_prefix: string
        invoice_settings: object
        livemode: boolean
        metadata: object
        name: string | null
        next_invoice_sequence: integer
        phone: string | null
        preferred_locales: array<string>
        shipping: object | null
        tax_exempt: "none" | "exempt" | "reverse"
Notes:
  - Use HTTPS only.
  - Never expose secret keys in client-side code.
  - Include the secret key server-side via Authorization header: "Authorization: Bearer <sk_test_xxx>".

Chain-of-thought

You are documenting an API. Think silently through these steps:
1. Identify purpose of the endpoint.
2. Extract headers, request fields, and response fields.
3. Identify all errors or missing fields.
4. Plan how to structure the output cleanly.

Then OUTPUT ONLY the final documentation with these sections:
Title
Description
Endpoint & Method
Headers
Parameters (Markdown table)
Example Request
Example Response
Errors
Notes / Best Practices

Rules:
- If a field or error is not present in the spec, write: "Not specified in the spec."
- Do NOT invent information or endpoints.
- Limit total length to 500 words.

API Specification:
Endpoint: POST /v1/customers
Description: Create a new customer object.
Headers:
  - Authorization: Bearer <sk_test_xxx> (required)
  - Content-Type: application/json (required)
Request Body (JSON, optional):
  - metadata: object (optional) — arbitrary key/value pairs. Example: {"order_id":"6735"}
Responses:
  - 200 OK:
      body (example excerpt):
        id: "cus_123456789" (string)
        object: "customer" (string)
        address: object | null
        balance: integer
        created: integer (unix timestamp)
        currency: string | null
        default_source: string | null
        delinquent: boolean
        description: string | null
        discount: object | null
        email: string | null
        invoice_prefix: string
        invoice_settings: object

livemode: boolean
metadata: object
name: string | null
next_invoice_sequence: integer
phone: string | null
preferred_locales: array<string>
shipping: object | null
tax_exempt: "none" | "exempt" | "reverse"

Notes:
- Use HTTPS only.
- Never expose secret keys in client-side code.
- Include the secret key server-side via Authorization header: "Authorization: Bearer <sk_test_xxx>".