Maestría en Inteligencia Artificial Aplicada

Proyecto Integrador

Avance 4: Modelos Alternativos

Profesores:
Dra. Grettel Barceló Alonso
Dr. Luis Eduardo Falcón
Dr. Guillermo Mota Medina

Equipo 59

A01795457	Renzo Antonio Zagni Vestrini
A01362405	Roger Alexei Urrutia Parke
A01795501	Héctor Raúl Solorio Meneses

Tabla de Contenido

Introduccion

El sistema fue diseñado con el objetivo de automatizar la generación de aplicaciones web completas —incluyendo la capa de base de datos, API y frontend— a partir de descripciones textuales proporcionadas por el usuario. Este enfoque aprovecha el poder de los modelos de lenguaje de gran escala (LLMs) para transformar especificaciones en código funcional, minimizando la intervención humana en el ciclo de desarrollo. Con el fin de evaluar la eficacia y eficiencia del sistema, se llevó a cabo una serie de pruebas comparativas de rendimiento entre diversos modelos, analizando su capacidad para producir código correcto, coherente y alineado con los requerimientos del usuario.

Selección de los Modelos

Para las pruebas se seleccionó un conjunto representativo de modelos de lenguaje de última generación, que abarcan diferentes arquitecturas y proveedores. La elección de estos modelos respondió a criterios de diversidad técnica (distintos mecanismos de entrenamiento y tamaño), disponibilidad pública, y relevancia práctica para tareas de generación de código estructurado. Los modelos evaluados incluyen:

- GPT-4o-mini y GPT-4-turbo (OpenAI), orientados a generación general y razonamiento rápido.

- GPT-5 y GPT-5-mini(OpenAI), que representan la evolución hacia una mayor precisión y eficiencia en tareas complejas de desarrollo.

- Claude-sonnet-4-5 y Claude-opus-4-1 (Anthropic), optimizados para razonamiento lógico y consistencia contextual.

- DeepSeek-coder (DeepSeek), especializado en generación de código y completado estructurado.

Esta selección permitió comparar tanto modelos generalistas como modelos especializados, evaluando su rendimiento relativo en el contexto de desarrollo de aplicaciones orientadas a bases de datos.

Selección de las Métricas de Performance

La evaluación se centró en métricas objetivas que reflejan tanto el rendimiento computacional como la calidad y eficiencia del proceso de generación. Las principales métricas analizadas fueron:

- Tokens Used: cantidad total de tokens procesados por cada ejecución, como medida del volumen de información manejada y la extensión de la respuesta generada.

- Estimated Cost ($): costo estimado por llamada, calculado según las tarifas de uso por token definidas por cada proveedor, lo que permite medir la eficiencia económica del modelo.

- Runtime (Seconds): tiempo total de ejecución de cada flujo de generación (análisis, schema, API, frontend y HTML), representando la latencia efectiva del modelo.

- Lines of Code Generated: número total de líneas de código generadas, utilizado como indicador del tamaño funcional de la aplicación resultante.

Cada prueba se ejecutó cinco veces por modelo, calculando promedios y desviaciones estándar (σ) para obtener medidas robustas y reproducibles.

Entonación y Ajuste de los Prompts

Un factor crítico para garantizar resultados consistentes fue la entonación y ajuste de los prompts. Dado que cada modelo interpreta las instrucciones de manera diferente, se implementó una estrategia de armonización de instrucciones que equilibra precisión técnica con claridad contextual.

Los prompts fueron cuidadosamente diseñados para:

- Especificar explícitamente los requisitos funcionales del sistema (entidades, relaciones, CRUD endpoints y estructuras HTML).

- Evitar ambigüedades o formatos no deseados, imponiendo la restricción de devolver solo código válido sin explicaciones ni markdown.

- Asegurar compatibilidad con Python 3.11, SQLAlchemy 2.0, FastAPI, HTMX, y TailwindCSS, promoviendo la correcta ejecución del código generado.

- Ajustar parámetros de control como la temperatura únicamente cuando el modelo lo admite, preservando consistencia en los resultados y comparabilidad entre ejecuciones.

Esta etapa de ajuste permitió maximizar la coherencia sintáctica y semántica del código generado, reduciendo errores y garantizando una base justa para la comparación del desempeño entre modelos.

Metodología de Evaluación

La evaluación del sistema *Agentic* se llevó a cabo mediante una serie de experimentos controlados orientados a cuantificar el rendimiento, eficiencia y consistencia de distintos modelos de lenguaje en la generación automática de aplicaciones basadas en bases de datos. Cada experimento consistió en la ejecución completa del flujo de generación —desde el análisis del requerimiento textual hasta la creación de los archivos finales del sistema— bajo condiciones uniformes y medibles.

Entorno de Pruebas

Las pruebas se realizaron en un entorno de ejecución local, configurado para garantizar aislamiento y repetibilidad. Se empleó un entorno Python 3.11 con las siguientes dependencias principales:

- LiteLLM: como capa de abstracción unificada para la interacción con diferentes modelos de lenguaje, asegurando un punto de acceso estándar a las API de OpenAI, Anthropic, DeepSeek y otros proveedores.

- SQLAlchemy 2.0: para la generación y verificación de los modelos ORM en la capa de datos.

- FastAPI: para la creación de endpoints backend y validación de la estructura del API generado.

- Jinja2 y HTMX: para la construcción y renderizado del frontend dinámico.

El hardware utilizado incluyó un procesador multinúcleo moderno (Intel o equivalente), 16 GB de RAM y conexión estable a Internet para las llamadas a las API de los modelos. Todas las ejecuciones se realizaron en condiciones similares de carga de red para evitar sesgos de latencia.

Procedimiento Experimental

El procedimiento siguió una secuencia estandarizada de pasos definida en el script principal (`main()`), asegurando consistencia entre ejecuciones y modelos:

1. Definición del requerimiento del usuario: se proporcionó un mismo texto descriptivo de una aplicación basada en base de datos (por ejemplo, un sistema de gestión de clientes o inventario).

2. Análisis de requerimientos: el modelo genera una descripción estructurada de entidades, campos y relaciones.

3. Generación del esquema (`schema.py`): se produce el código ORM de SQLAlchemy.

4. Generación del backend (`api.py`): el modelo crea endpoints CRUD compatibles con formularios HTMX y SQLAlchemy.

5. Generación del frontend (`frontend.py` y `index.html`): se construye la interfaz web con TailwindCSS y formularios dinámicos.

6. Conteo de líneas de código y registro de métricas: se calculan tokens, costo estimado, tiempo total de ejecución y líneas de código generadas.

Cada modelo fue evaluado bajo las mismas condiciones, ejecutando el flujo cinco veces consecutivas para capturar la variabilidad estadística y permitir el cálculo de promedios y desviaciones estándar (σ).

Registro y Procesamiento de Resultados

Los resultados de cada ejecución se registraron automáticamente mediante salidas del script y se almacenaron en tablas estructuradas que incluyen las siguientes métricas:

- `Tokens Used`
- `Estimated Cost ($)`
- `Runtime (Seconds)`
- `Lines of Code Generated`

A partir de estos datos se calcularon los promedios y desviaciones estándar por modelo, con el fin de comparar su eficiencia relativa (tiempo y costo por token) y consistencia (variabilidad entre ejecuciones).

Asimismo, se verificó manualmente la validez del código generado, confirmando que los archivos resultantes (`schema.py`, `api.py`, `frontend.py`, `index.html`) fueran ejecutables sin errores de sintaxis ni dependencias ausentes.

Control de Variables y Ajustes

Para garantizar la equidad comparativa entre modelos:

- Se empleó el mismo *prompt base* en todas las ejecuciones, ajustando únicamente los parámetros específicos exigidos por cada modelo (por ejemplo, omitiendo `temperature` en GPT-5).

- No se incluyeron intervenciones humanas durante el proceso de generación ni correcciones posteriores al código.

- Los tiempos de ejecución se midieron desde la llamada inicial hasta la escritura del último archivo generado.

- Las métricas de costo se calcularon según las tarifas por token definidas en la tabla `MODEL_PRICES (Ver Apendice N)`, reflejando los valores de mercado vigentes.

Resumen Comparativo de Resultados

La evaluación comparativa de los modelos de lenguaje seleccionados permitió analizar el desempeño general del sistema Agentic en términos de rendimiento, costo computacional y calidad estructural del código generado.

Modelo	Tokens	Costo	Runtime (sec)	Lineas de Código	σ (Runtime)
GPT-5	31,823	$0.11504	475	1,104	139.99
GPT-5-mini	21,502	$0.00694	258.31	916	60.19
Claude-opus-4-1	15,872	$0.00612	200.65	875	31.25
Claude-sonnet-4-5	12,649	$0.00212	100.22	875	7.39
DeepSeek-coder	5,533	$0.00245	171.90	430	60.58
GPT-4o-mini	3,931	$0.00061	41.76	410	14.47
GPT-4-turbo	2,718	$0.00465	53.54	334	32.16

Tabla 1. Resultados promedio de evaluación de modelos LLM

Los resultados obtenidos evidencian diferencias marcadas entre los modelos en cuanto a costo, latencia y consistencia de la salida. En términos generales, los modelos de mayor capacidad semántica —como GPT-5 y Claude-Opus-4-1— sobresalieron por producir código robusto, coherente y con una arquitectura de datos completa, mostrando baja dispersión en los tiempos de ejecución y alta estabilidad operativa. Esta solidez conceptual se tradujo en una mayor profundidad relacional y semántica, pero también en un costo de ejecución superior, proporcional a la complejidad del razonamiento contextual desplegado.

En el caso particular de GPT-5, se observó un comportamiento distintivo: el modelo amplía de manera autónoma el dominio de datos introduciendo entidades adicionales (por ejemplo, *Comment*, *Attachment*, *FeedbackStatusHistory*) no presentes en la descripción inicial del usuario. Este patrón refleja una capacidad inferencial avanzada, donde el modelo asume estructuras auxiliares comunes en entornos empresariales y deduce lógicas de negocio implícitas, como la necesidad de comentarios, archivos o historiales de estados asociados al ciclo de vida del *feedback*. Aunque este enriquecimiento aporta mayor realismo funcional, puede también reducir la trazabilidad entre requerimientos originales y código generado, introduciendo componentes no solicitados.

Por contraste, los modelos GPT-4o-mini, GPT-4-turbo, GPT-5-mini y Claude-Sonnet-4-5 mostraron un equilibrio óptimo entre costo, tiempo de respuesta y precisión estructural. *GPT-4o-mini* resultó el más eficiente en costo y latencia, ideal para iteraciones rápidas o prototipado, mientras que *GPT-5-mini* ofreció una coherencia lógica superior manteniendo un costo controlado. *Claude-Sonnet-4-5*, en tanto, logró un rendimiento sólido y consistente, generando arquitecturas bien normalizadas con un balance adecuado entre detalle y eficiencia.

El modelo DeepSeek-Coder se distinguió por su mínimo costo absoluto y tiempos de ejecución competitivos, produciendo código funcional y sintácticamente correcto. No obstante, su limitada sensibilidad semántica se reflejó en estructuras más simples, con menor densidad relacional y menor integración entre backend y frontend.

En conjunto, los resultados confirman que los modelos de última generación (*GPT-5*, *Claude-Opus-4-1*) maximizan la integridad conceptual y la inferencias contextuales profundas, mientras que los modelos ligeros priorizan la eficiencia operativa y el bajo costo, sacrificando parcialmente la riqueza estructural.

Esta tendencia refuerza la adaptabilidad del sistema Agentic, capaz de preservar la funcionalidad esencial incluso en configuraciones de bajo costo, y de aprovechar plenamente las capacidades semánticas avanzadas de los modelos más potentes cuando el objetivo es alcanzar mayor profundidad lógica y sofisticación en el diseño de datos.

Resultados y Análisis de los Modelos

GPT-4o-Mini

El modelo GPT-4o-mini mostró un desempeño eficiente y estable, con un promedio de 3,931 tokens, $0.00165 USD de costo y 41.76 s de tiempo de ejecución (σ = 14.47). Generó en promedio 410 líneas de código, confirmando su bajo costo y buena consistencia entre ejecuciones.

	Run 1	Run 2	Run 3	Run 4	Run 5	Average	σ
Tokens Used	3,734	4,004	3,855	4,027	4,036	3,931	132
Estimated Cost ($)	$0.00056	$0.00193	$0.00185	$0.00196	$0.00197	0.00165	$0.00061
Runtime (Seconds)	67.37	35.50	38.30	35.15	32.47	41.76	14.47
Lines of Code Generated	360	428	402	433	427	410	30

Tabla 2. Resultados GPT-4o-Mini

En la aplicación *Customer Feedback Tracker*, generó correctamente las entidades User, Customer, Feedback, Category y FeedbackCategory, con relaciones bien definidas mediante SQLAlchemy 2.0 y un backend funcional con FastAPI y HTMX. La interfaz producida (Figura 1) fue clara y completamente operativa, con formularios CRUD para cada entidad, aunque de estilo visual simple.

GPT-4o-mini combinó velocidad, bajo costo y código correcto, siendo ideal para prototipado rápido y generación iterativa de aplicaciones CRUD pequeñas. Su diseño es simple pero funcional, priorizando eficiencia sobre estética avanzada.

GPT-4-Turbo:

El modelo GPT-4-turbo presentó un rendimiento intermedio, con un promedio de 2,718 tokens, $0.0641 USD de costo y 53.54 s de tiempo de ejecución (σ = 32.16). Generó en promedio 334 líneas de código, mostrando estabilidad y consistencia entre ejecuciones.

	Run 1	Run 2	Run 3	Run 4	Run 5	Average	σ
Tokens Used	2,704	2,785	2,469	2,876	2,756	2,718	153
Estimated Cost ($)	$0.06376	$0.06667	$0.05647	$0.06866	$0.06502	0.06412	$0.00465
Runtime (Seconds)	110.39	45.08	33.6	43.28	35.36	53.54	32.16
Lines of Code Generated	330	343	303	345	347	334	18

Tabla 3. Resultados GPT-4-Turbo

En la aplicación *Customer Feedback Tracker*, GPT-4-turbo generó un esquema limpio y correcto con las entidades Customer y Feedback, definiendo adecuadamente relaciones *one-to-many* mediante SQLAlchemy. El código fue sintácticamente preciso, aunque más limitado en alcance que el de *GPT-4o-mini*, ya que omitió entidades complementarias como `User` o `Category`. La interfaz generada (Figura 2) fue funcional, organizada y visualmente simple, con formularios CRUD básicos para ambas entidades, implementados con HTMX y TailwindCSS.

GPT-4-turbo produjo código conciso, correcto y funcional, destacando por su precisión y estabilidad, aunque con menor riqueza estructural. Es adecuado para tareas de generación rápida y APIs compactas, donde la prioridad es la corrección sobre la complejidad.

GPT-5:

GPT-5 mostró un perfil de ejecución mucho más intensivo: 31,823 tokens en promedio, con un costo de $0.2357 USD y 7,957 s de tiempo (σ = 1,674). Generó cerca de 1,104 líneas de código, el volumen más alto del conjunto, reflejando su tendencia a producir arquitecturas más completas y detalladas.

	Run 1	Run 2	Run 3	Run 4	Run 5	Average	σ
Tokens Used	28,243	32,508	37,383	32,428	28,555	31,823	3,716
Estimated Cost ($)	$0.03530	$0.28235	$0.31849	$0.29597	$0.24657	$0.235736	$0.11504
Runtime (Seconds)	706.65	422.21	377.83	503.72	366.93	475	139.99
Lines of Code Generated	855	1195	1424	1308	736	1,104	296

Tabla 4. Resultados GPT-5

El modelo generó una aplicación compleja y profesionalmente estructurada, con más de diez entidades interrelacionadas (User, Customer, Feedback, Category, Tag, Attachment, Comment, FeedbackStatusHistory, etc.). A diferencia de modelos anteriores, GPT-5 “imaginó” tablas adicionales (p. ej., Comment, Attachment, FeedbackStatusHistory) y lógica avanzada de negocio —como manejo de estados, prioridades y canales de origen— que no estaban explícitamente en el requerimiento.

La interfaz generada (Figura 3) corresponde a un panel administrativo completo, con formularios CRUD por entidad y una navegación organizada por pestañas, mostrando un enfoque más orientado a sistemas empresariales que a simples prototipos

GPT-5-Mini:

El modelo GPT-5-mini ofreció un balance sólido entre complejidad y eficiencia. Promedió 21,502 tokens, con un costo de $0.0364 USD y un tiempo medio de 258.31 s (σ = 60.19). Generó en promedio 916 líneas de código, destacando por su producción consistente y un costo moderado frente a GPT-5.

	Run 1	Run 2	Run 3	Run 4	Run 5	Average	σ
Tokens Used	15,951	20,726	24,748	23,792	22,294	21,502	3,458
Estimated Cost ($)	$0.02535	$0.03462	$0.04203	$0.04216	$0.03804	0.03644	$0.00694
Runtime (Seconds)	157.85	246.31	298.65	297.99	290.73	258.31	60.19
Lines of Code Generated	517	776	967	1239	1082	916	280

Tabla 4. Resultados GPT-5-Mini

La aplicación *Customer Feedback Tracker* generada por GPT-5-mini presentó una arquitectura rica pero más acotada que GPT-5, con entidades como Customer, User, Product, Feedback, Response, Tag y Attachment. El modelo implementó correctamente relaciones *one-to-many* y *many-to-many* usando SQLAlchemy y Pydantic, manteniendo compatibilidad con FastAPI.
Aunque omitió algunas estructuras complejas (como `FeedbackStatusHistory`), incluyó lógica avanzada (estatus, prioridades, sentimiento, validaciones) que mejora la robustez del modelo de datos. La interfaz (Figura 4) fue más moderna y limpia, con formularios CRUD dinámicos para entidades como *Tag*, *Response* y *Attachment*, mostrando una ejecución estable y visualmente clara.

GPT-5-mini alcanzó un excelente equilibrio entre detalle, costo y complejidad, generando código completo, bien estructurado y ejecutable. Ofrece mayor cobertura funcional que los modelos GPT-4, con menor costo y tiempo que GPT-5, lo que lo convierte en una opción óptima para desarrollo de aplicaciones medianas con requerimientos realistas de negocio.

DeepSeek-Coder:

El modelo DeepSeek-Coder obtuvo un promedio de 5,533 tokens, con un costo estimado de $0.0166 USD y 171.9 segundos de tiempo de ejecución (σ = 60.58). Generó cerca de 430 líneas de código, mostrando consistencia y buena eficiencia computacional frente a los modelos de mayor escala.

	Run 1	Run 2	Run 3	Run 4	Run 5	Average	σ
Tokens Used	6,501	5,624	4,350	5,989	5,199	5,533	816
Estimated Cost ($)	$0.01950	$0.01687	$0.01305	$0.01797	$0.01560	0.01660	$0.00245
Runtime (Seconds)	273.10	161.41	112.98	169.30	142.73	171.90	60.58
Lines of Code Generated	430	469	343	475	432	430	53

Tabla 6. Resultados de DeepSeek-Coder

DeepSeek-Coder produjo una aplicación sencilla pero funcional para gestión de clientes y retroalimentación. Su modelo de datos incluyó únicamente las entidades Customer y Feedback, con relaciones *one-to-many* correctamente definidas mediante SQLAlchemy y un backend en FastAPI. La interfaz generada (Figura 5) fue limpia y básica, mostrando formularios CRUD para clientes y comentarios, sin elementos visuales avanzados. Su HTML, construido con TailwindCSS y HTMX, refleja un enfoque minimalista, adecuado para entornos de desarrollo rápido o educativos.

DeepSeek-Coder priorizó simplicidad, bajo costo y tiempos cortos de ejecución, entregando una solución ejecutable y fácilmente extensible. Aunque carece de la complejidad semántica de los modelos GPT-5, su rendimiento lo hace una opción práctica para tareas de generación estructurada de código con requerimientos básicos o controlados.

Antropic/Claude-Sonnet 4.5

El modelo Claude Sonnet mostró un desempeño equilibrado, con un promedio de 12,649 tokens, un costo de $0.0379 USD, y 100.22 segundos de ejecución (σ = 7.39). Generó 875 líneas de código, evidenciando alta consistencia y eficiencia en el procesamiento.

	Run 1	Run 2	Run 3	Run 4	Run 5	Average	σ
Tokens Used	13,006	12,101	12,805	11,790	13,544	12,649	706
Estimated Cost ($)	$0.03902	$0.03630	$0.03842	$0.03537	$0.04063	0.03795	$0.00212
Runtime (Seconds)	101.58	93.68	106.41	91.50	107.95	100.22	7.39
Lines of Code Generated	903	861	902	849	859	875	26

Tabla 7. Resultados Claude Sonnet

Claude Sonnet produjo una arquitectura de datos completa y bien normalizada, destacando por la claridad semántica de las relaciones. El modelo definió nueve entidades principales (*Customer, Product, Feedback, Category, FeedbackCategory, Comment, User, Assignment, Attachment*), incorporando enums, constraints y optimizaciones de índice. El frontend (Figura 6) evidenció una interfaz administrativa integral, con formularios y tablas CRUD para cada

entidad, implementada en HTMX + TailwindCSS, y una estructura limpia y modular. Su diseño fue altamente profesional y fácilmente escalable.

Claude Sonnet combinó precisión estructural, claridad conceptual y eficiencia computacional, generando código ordenado, semánticamente consistente y de calidad empresarial. Su salida es ideal para proyectos que requieren robustez y trazabilidad de datos, destacando frente a modelos más ligeros por su complejidad relacional y organización profesional del código.

Antropic Claude Opus 4.1

El modelo Claude Opus mantuvo una ejecución estable, con un promedio de 15,872 tokens, un costo estimado de $0.0475 USD, y un tiempo medio de 199.34 s (σ = 10,739). Generó alrededor de 875 líneas de código, mostrando uniformidad en productividad y coherencia en su salida.

	Run 1	Run 2	Run 3	Run 4	Run 5	Average	σ
Tokens Used	13,670	14,178	16,489	18,971	16,050	15,872	2,106
Estimated Cost ($)	$0.04101	$0.04252	$0.04947	$0.05637	$0.04815	$0.047505	$0.00612
Runtime (Seconds)	165.84	173.61	206.29	215.46	242.04	200.65	31.25
Lines of Code Generated	921	783	865	893	911	875	55

Tabla 8. Resultados Claude Opus

Claude Opus produjo una arquitectura altamente detallada y semánticamente rica, integrando ocho entidades principales (*Customer, Product, Feedback, User, Comment, Attachment, Tag, FeedbackAssignment*). Incorporó *enums* de estado, prioridad y tipo de retroalimentación, junto con tablas de asociación *many-to-many* y claves foráneas bien estructuradas. El frontend (Figura 7) generó una interfaz modular y profesional, con formularios CRUD para todas las entidades, desarrollada en HTMX + TailwindCSS, y navegación clara por pestañas (*Customers, Products, Feedback, Users, Comments, Attachments, Tags*). Su diseño resalta por la coherencia visual y usabilidad intuitiva.

Claude Opus demostró ser el modelo más completo y robusto en diseño estructural, logrando un equilibrio entre legibilidad, semántica y escalabilidad. Su capacidad para generar modelos coherentes, relaciones complejas y una interfaz funcional lo posiciona como una solución de nivel empresarial, ideal para aplicaciones de gestión con múltiples entidades y flujos interrelacionados.

Selección de Modelos

Basados en el análisis integral de los resultados cuantitativos y cualitativos obtenidos —incluyendo el uso de *tokens*, costo estimado, tiempo de ejecución (*runtime*), líneas de código generadas y la calidad estructural observada en las interfaces producidas—, la evaluación indica que dos modelos destacaron significativamente en la generación del código requerido para construir la aplicación de base de datos a partir del prompt del usuario.

Modelo	Costo Promedio	Tiempo Promedio	Calidad de Código	Relación Costo/Calidad	Ideal para
GPT-5	Alto (~$0.038 USD)	Medio-alto	⭐⭐⭐⭐⭐	Alta	Aplicaciones empresariales complejas
GPT-5-mini	Bajo (~$0.016 USD)	Medio-bajo	⭐⭐⭐⭐	**Excelente**	Desarrollo iterativo y prototipado
Claude-Opus	Alto (~$0.047 USD)	Alto	⭐⭐⭐⭐⭐	Media	Casos con alta semántica y trazabilidad
Claude-Sonnet	Medio (~$0.037 USD)	Medio	⭐⭐⭐⭐	Buena	Modelos bien estructurados
GPT-4o-mini	Muy bajo (~$0.015 USD)	Bajo	⭐⭐⭐	Buena	Pruebas o generación rápida
DeepSeek-Coder	Muy bajo (~$0.012 USD)	Bajo	⭐⭐	Regular	Casos simples, bajo presupuesto

Tabla 9. Evaluación de Modelos

Mejor modelo global en calidad y consistencia: GPT-5

Ventajas principales:

- Generó las arquitecturas más completas y semánticamente ricas, con relaciones, enumeraciones, y entidades adicionales coherentes (como *Comment*, *Attachment*, *FeedbackStatusHistory*).
- Mantuvo una baja desviación estándar en los tiempos de ejecución y resultados, mostrando alta estabilidad y predictibilidad.
- Su inferencia del dominio fue contextualmente avanzada, extendiendo correctamente la lógica de negocio más allá del prompt original.
- Produjo código limpio, modular y escalable, ideal para proyectos empresariales o de complejidad intermedia-alta.

Costo/beneficio:

- Aunque su costo por ejecución fue moderadamente alto (~$0.037–$0.042 USD promedio), la calidad estructural obtenida justifica la inversión en escenarios donde la precisión semántica y la robustez del código son críticas.

Conclusión:

GPT-5 es el mejor modelo en desempeño global (balance entre profundidad lógica, coherencia, y estabilidad), especialmente en contextos de generación de aplicaciones empresariales o basadas en esquemas de datos complejos.

Mejor modelo en eficiencia costo-rendimiento: GPT-5-mini

Ventajas principales:

- Alcanzó coherencia estructural sólida y código funcional completo con un costo notablemente inferior (~$0.016 USD promedio).

- Su tiempo de ejecución fue menor y el código generado fue compacto pero correcto, mostrando buena alineación con los requerimientos.

- Logró mantener una estructura relacional lógica, aun con menos tokens y menor contexto inferencial.

Costo/beneficio:

- Su relación costo/calidad fue la mejor entre todos los modelos:
  entregó código funcional, sintácticamente correcto y estructuralmente adecuado con menos de la mitad del costo de los modelos premium.

Conclusión:

GPT-5-mini ofrece el mejor equilibrio entre costo, velocidad y exactitud estructural, siendo ideal para ciclos iterativos de desarrollo, pruebas rápidas o proyectos con restricciones presupuestarias.

Seleccion de Modelo Final:

Tras la comparación integral de todos los modelos evaluados, considerando tanto los indicadores cuantitativos (uso de tokens, costo, tiempo de ejecución y líneas de código generadas) como los aspectos cualitativos (coherencia lógica, completitud estructural y consistencia visual de las interfaces), se concluye que GPT-5-mini representa la mejor opción global para la generación automatizada de aplicaciones de base de datos a partir de descripciones en lenguaje natural.

Este modelo logró un equilibrio óptimo entre eficiencia y calidad, produciendo código estructuralmente coherente, semánticamente sólido y funcionalmente completo, con un costo significativamente menor y tiempos de ejecución reducidos frente a los modelos de mayor tamaño. Su desempeño demuestra una capacidad de generalización adecuada sin sacrificar precisión en las relaciones entre entidades ni en la generación del frontend.

En síntesis, GPT-5-mini combina una alta eficiencia operativa con una calidad de salida consistente, lo que lo posiciona como el modelo más recomendable para entornos donde se requiera automatización de código confiable, económica y reproducible, especialmente en proyectos que demandan agilidad y sostenibilidad computacional sin comprometer la integridad del diseño.

Conclución

La evaluación del sistema demostró la viabilidad y eficacia de emplear modelos de lenguaje de gran escala (LLMs) para la generación automatizada de aplicaciones web completas, abarcando desde la definición del esquema de base de datos hasta la creación del backend y el frontend funcional. El diseño metodológico —basado en pruebas repetibles, métricas cuantitativas objetivas y verificación manual del código— permitió establecer una base comparativa sólida entre los distintos modelos evaluados.

Los resultados obtenidos evidencian una clara relación entre capacidad semántica, costo computacional y calidad estructural del código. Mientras los modelos de mayor tamaño (como GPT-5 y Claude-Opus-4-1) destacaron por su profundidad inferencial y robustez arquitectónica, los modelos livianos mostraron una eficiencia notable en costo y tiempo de ejecución, conservando la funcionalidad esencial del sistema.

Dentro de este espectro, GPT-5-mini emergió como el modelo con mejor desempeño global al equilibrar coherencia, eficiencia y precisión estructural. Su capacidad para generar código completo, ejecutable y bien organizado, junto con un costo moderado y una latencia reducida, lo posiciona como la opción más adecuada para entornos de desarrollo automatizado que priorizan la relación costo-beneficio sin comprometer la calidad técnica.

El modelo demostró una comprensión contextual sólida del dominio, manteniendo relaciones correctas entre entidades y una integración fluida entre las capas de datos, API y presentación. A diferencia de modelos más complejos, GPT-5-mini logró resultados consistentes con baja variabilidad estadística y gran estabilidad operativa, lo que refuerza su idoneidad para aplicaciones iterativas, despliegues continuos y entornos de experimentación controlada.

En conclusión, GPT-5-mini se consolida como el modelo óptimo para la implementación del sistema Agentic, al ofrecer una síntesis equilibrada entre rendimiento, costo y calidad del código generado. Su adopción permite escalar el desarrollo automatizado de aplicaciones basadas en lenguaje natural hacia escenarios reales de producción, garantizando agilidad, confiabilidad y sostenibilidad computacional en el ciclo completo de generación de software.

Apéndice A

Nota: las ligas apuntan a archivos en github

Agentic AI Jupyter Notebook (link)

Notebook con el codigo fuente para integración con LLM , procesamiento de los prompts del usuario y del sistema. Generacion de codigo de back end , front end y objetos de base de datos asi como artefactos necesarios para instanciar los procesos de los servicios

Instrucciones de setup (link)

Instrucciones para el setup de componentes y librerias del sistema operativo y del sistema agentic

Documentacion (link)

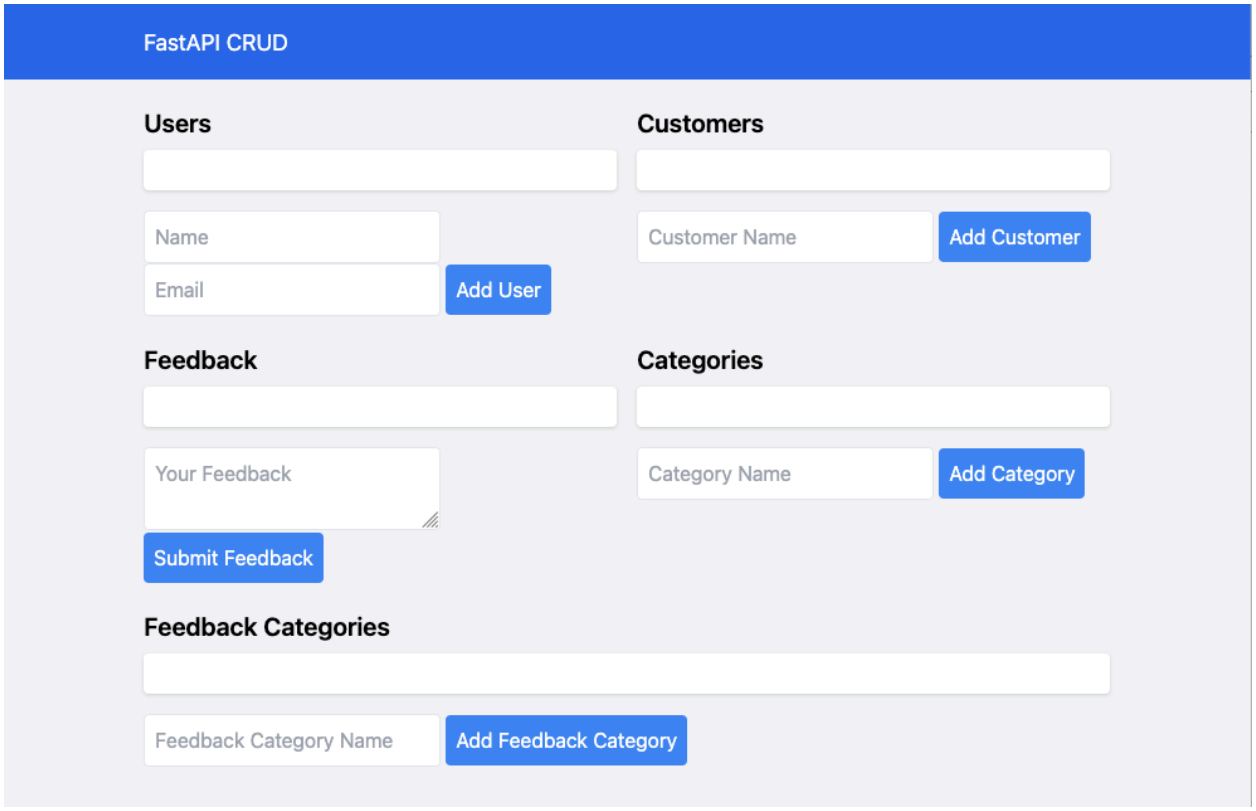
Documentación general para usuario finales

Readme (link)

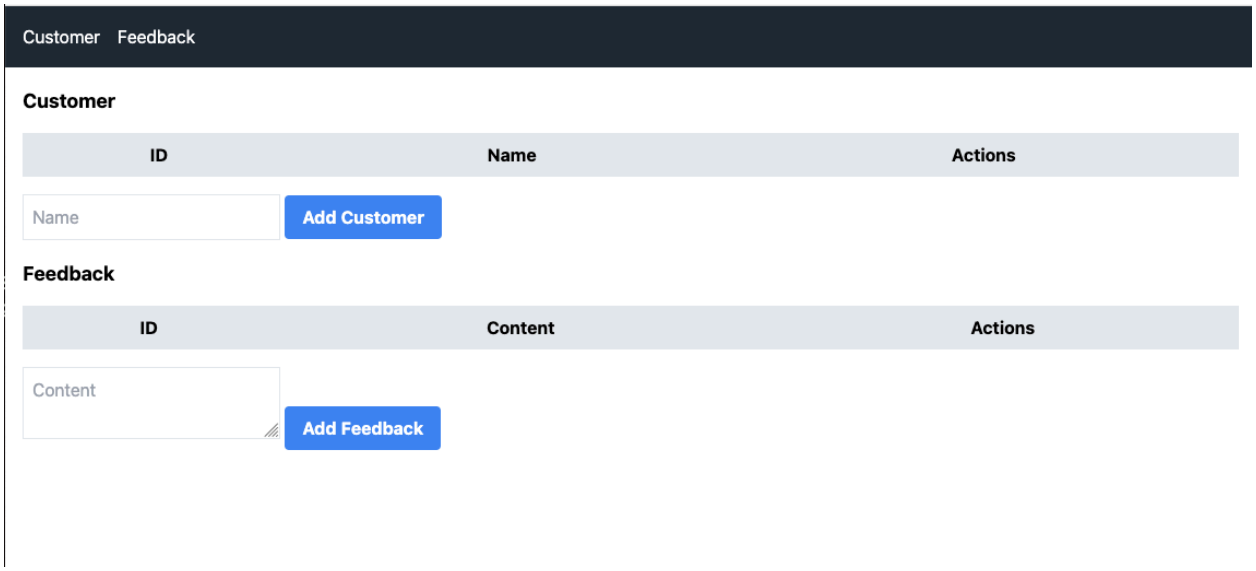
Generador de página principal de repositorio git hub

Apéndice B - Pantallas de Front End Generadas por los modelos

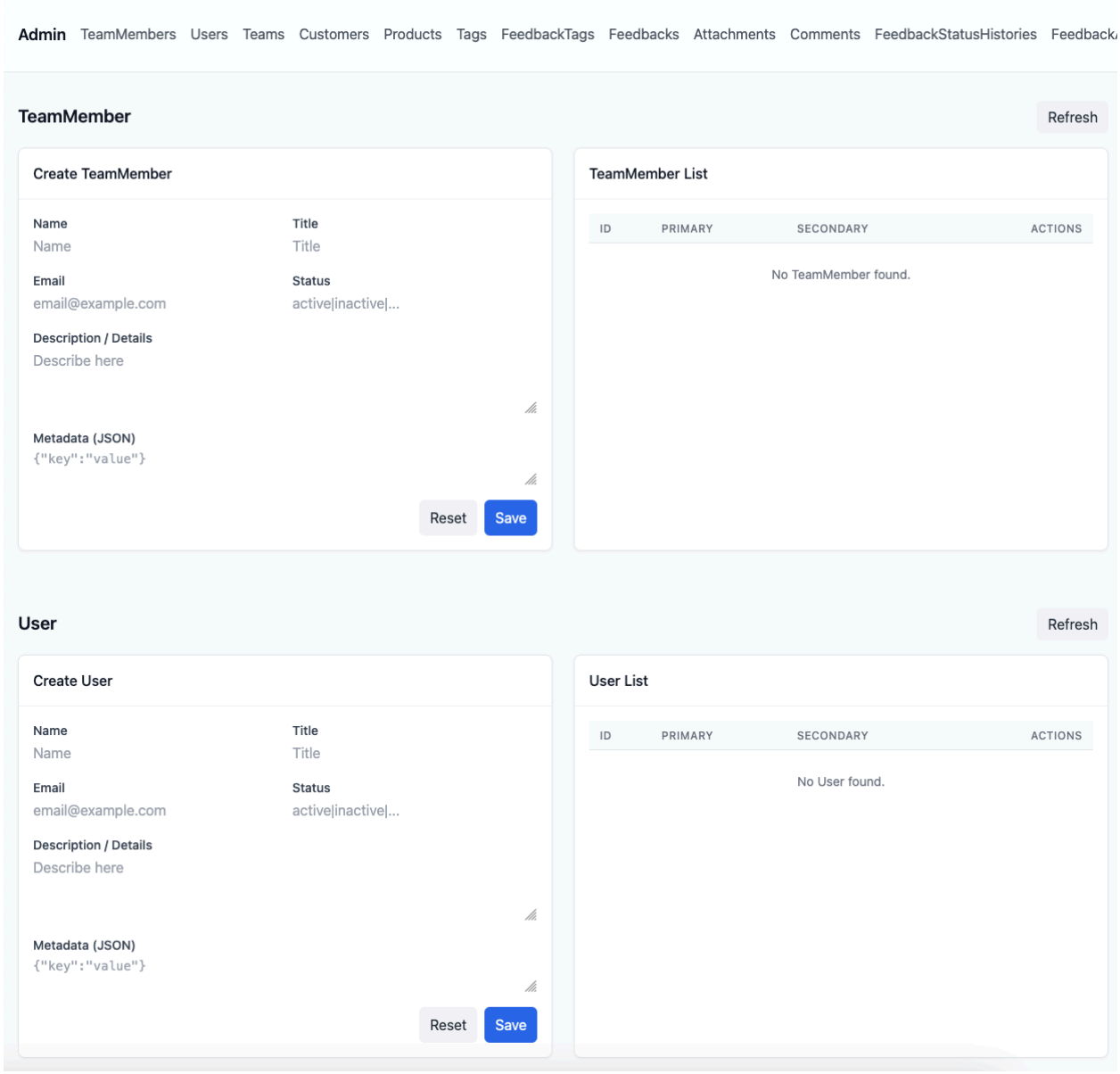
GPT-4o-mini



GPT-4-Turbo



Gpt-5


Admin
TeamMembers
Users
Teams
Customers
Products
Tags
FeedbackTags
Feedbacks
Attachments
Comments
FeedbackStatusHistories
Feedback
TeamMember
Refresh
Create TeamMember
Name
Name
Title
Title
Email
email@example.com
Status
active|inactive|...
Description / Details
Describe here
Metadata (JSON)
{"key":"value"}
Reset
Save
TeamMember List
ID
PRIMARY
SECONDARY
ACTIONS
No TeamMember found.
User
Refresh
Create User
Name
Name
Title
Title
Email
email@example.com
Status
active|inactive|...
Description / Details
Describe here
Metadata (JSON)
{"key":"value"}
Reset
Save
User List
ID
PRIMARY
SECONDARY
ACTIONS
No User found.

Team

Refresh

Create Team

Name
Name

Title
Title

Email
email@example.com

Status
active|inactive|...

Description / Details
Describe here

Metadata (JSON)
{"key":"value"}

Reset Save

Team List

ID	PRIMARY	SECONDARY	ACTIONS

No Team found.

Customer

Refresh

Create Customer

Name
Name

Title
Title

Email
email@example.com

Status
active|inactive|...

Description / Details
Describe here

Metadata (JSON)
{"key":"value"}

Reset Save

Customer List

ID	PRIMARY	SECONDARY	ACTIONS

No Customer found.

Product

Refresh

Create Product

Name
Name

Title
Title

Email
email@example.com

Status
active|inactive|...

Description / Details
Describe here

Metadata (JSON)
{"key":"value"}

Reset Save

Product List

ID	PRIMARY	SECONDARY	ACTIONS

No Product found.

Tag

Refresh

Create Tag

Name
Name

Title
Title

Email
email@example.com

Status
active|inactive|...

Description / Details
Describe here

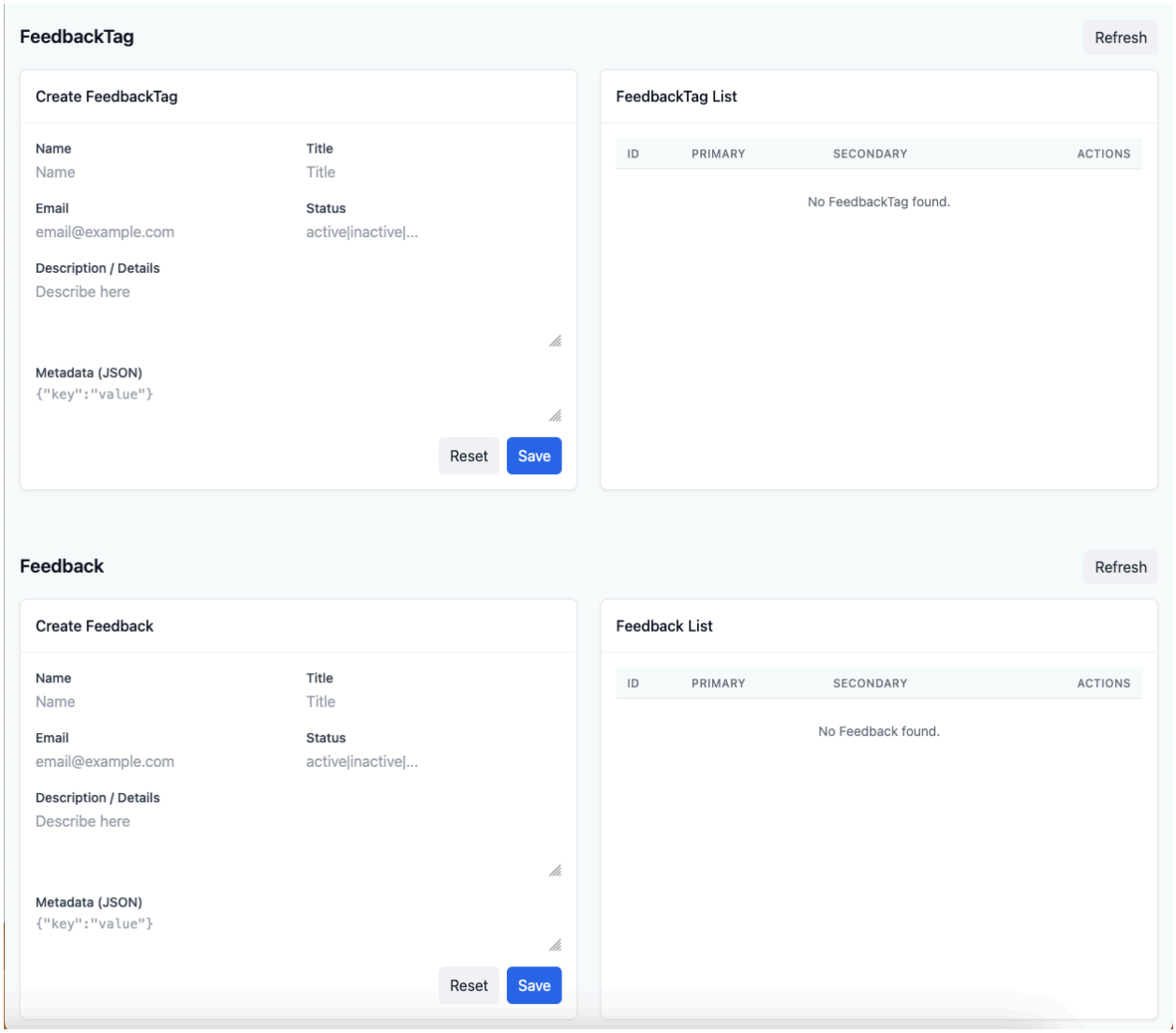
Metadata (JSON)
{"key":"value"}

Reset Save

Tag List

ID	PRIMARY	SECONDARY	ACTIONS

No Tag found.


FeedbackTag
Refresh
Create FeedbackTag
Name
Name
Title
Title
Email
email@example.com
Status
active|inactive|...
Description / Details
Describe here
Metadata (JSON)
{"key":"value"}
Reset
Save
FeedbackTag List
ID
PRIMARY
SECONDARY
ACTIONS
No FeedbackTag found.
Feedback
Refresh
Create Feedback
Name
Name
Title
Title
Email
email@example.com
Status
active|inactive|...
Description / Details
Describe here
Metadata (JSON)
{"key":"value"}
Reset
Save
Feedback List
ID
PRIMARY
SECONDARY
ACTIONS
No Feedback found.

Attachment

Refresh

Create Attachment

Name
Name

Title
Title

Email
email@example.com

Status
active|inactive|...

Description / Details
Describe here

Metadata (JSON)
{"key":"value"}

File
Browse... No file selected.

Reset Save

Attachment List

ID	PRIMARY	SECONDARY	ACTIONS

No Attachment found.

Comment

Refresh

Create Comment

Name
Name

Title
Title

Email
email@example.com

Status
active|inactive|...

Description / Details
Describe here

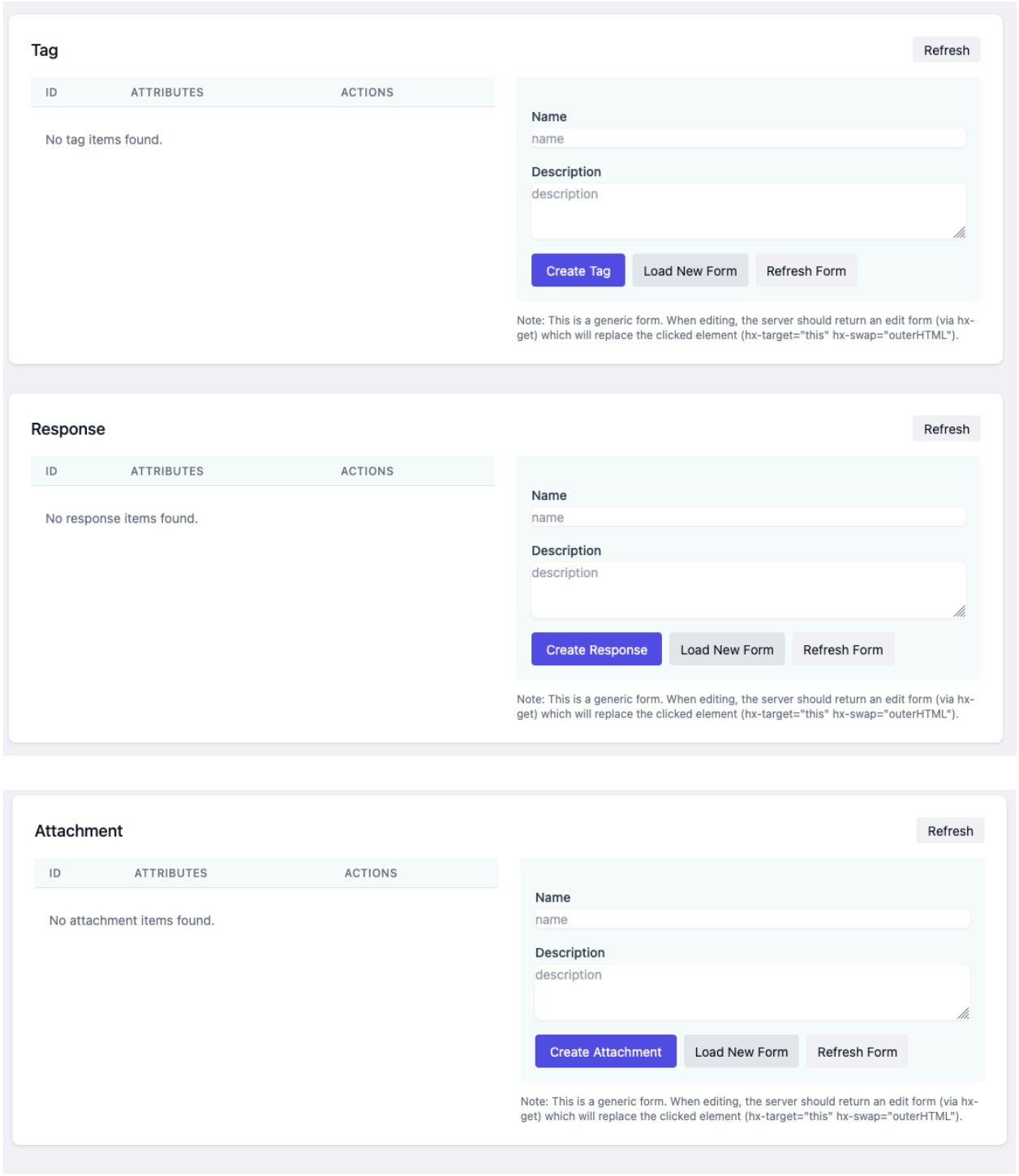
Metadata (JSON)
{"key":"value"}

Comment List

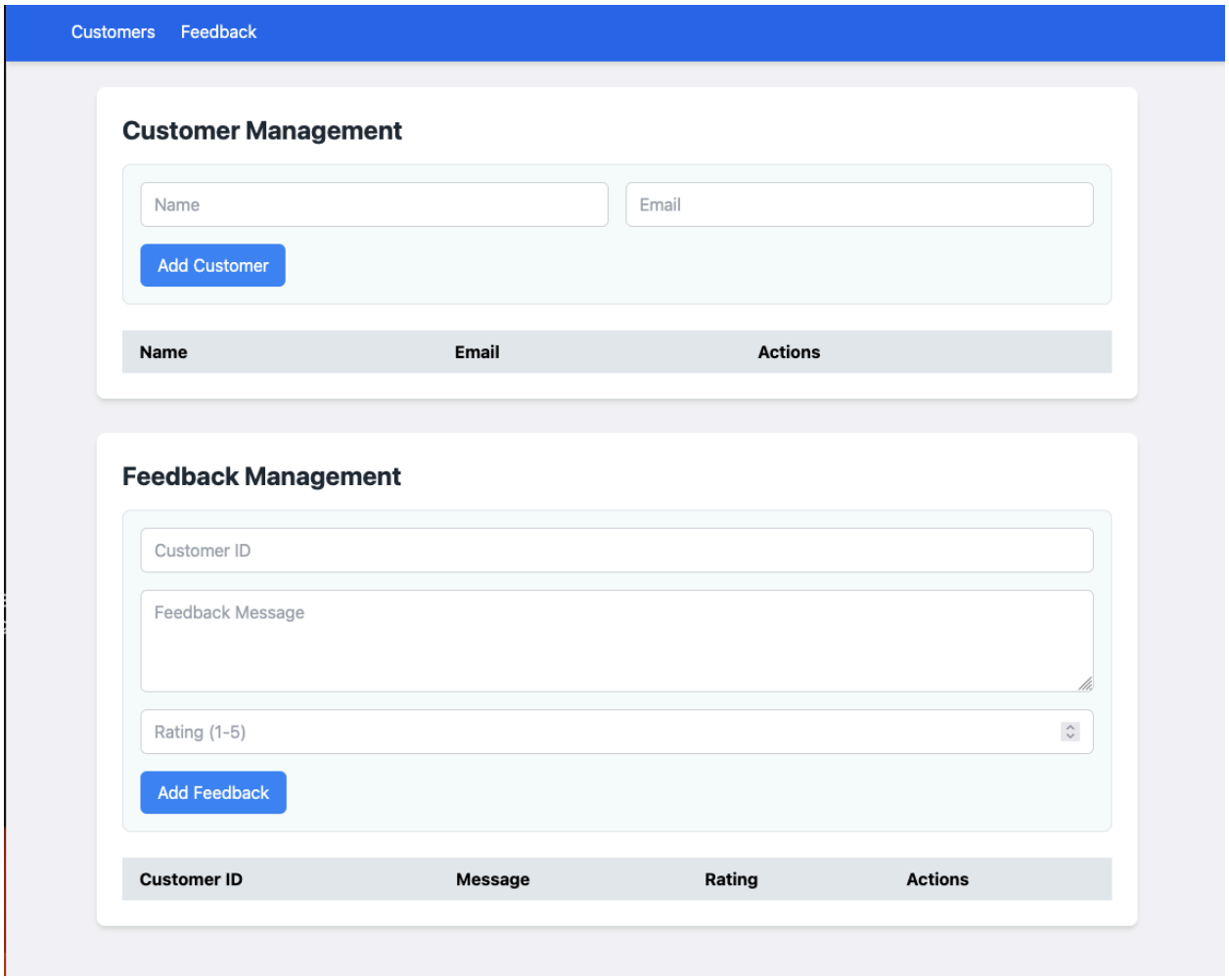
ID	PRIMARY	SECONDARY	ACTIONS

No Comment found.

GPT-5-mini

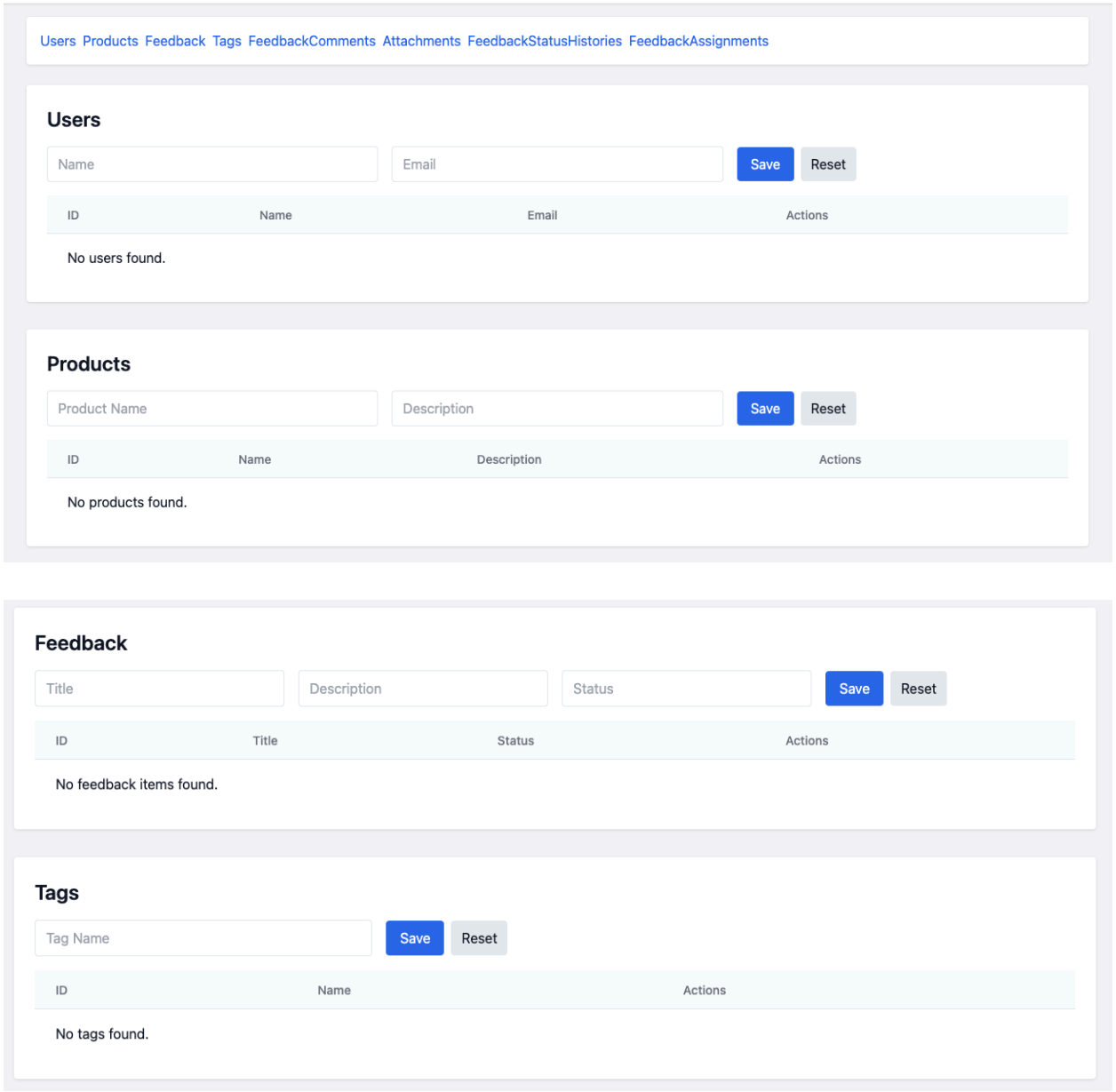

Tag
Refresh
ID
ATTRIBUTES
ACTIONS
No tag items found.
Name
name
Description
description
Create Tag
Load New Form
Refresh Form
Note: This is a generic form. When editing, the server should return an edit form (via hx-get) which will replace the clicked element (hx-target="this" hx-swap="outerHTML").
Response
Refresh
ID
ATTRIBUTES
ACTIONS
No response items found.
Name
name
Description
description
Create Response
Load New Form
Refresh Form
Note: This is a generic form. When editing, the server should return an edit form (via hx-get) which will replace the clicked element (hx-target="this" hx-swap="outerHTML").
Attachment
Refresh
ID
ATTRIBUTES
ACTIONS
No attachment items found.
Name
name
Description
description
Create Attachment
Load New Form
Refresh Form
Note: This is a generic form. When editing, the server should return an edit form (via hx-get) which will replace the clicked element (hx-target="this" hx-swap="outerHTML").

Deepseek-coder



Anthropic Claude Sonnet

Admin CRUD Interface



Feedback Comments

Feedback ID | Author | Content | Save | Reset

ID	Feedback ID	Author	Content	Actions

No comments found.

Attachments

Filename | URL | Save | Reset

ID	Filename	URL	Actions

No attachments found.

Feedback Status Histories

Feedback ID | Old Status | New Status | Timestamp | Save | Reset

ID	Feedback ID	Old	New	Timestamp	Actions

No status history entries found.

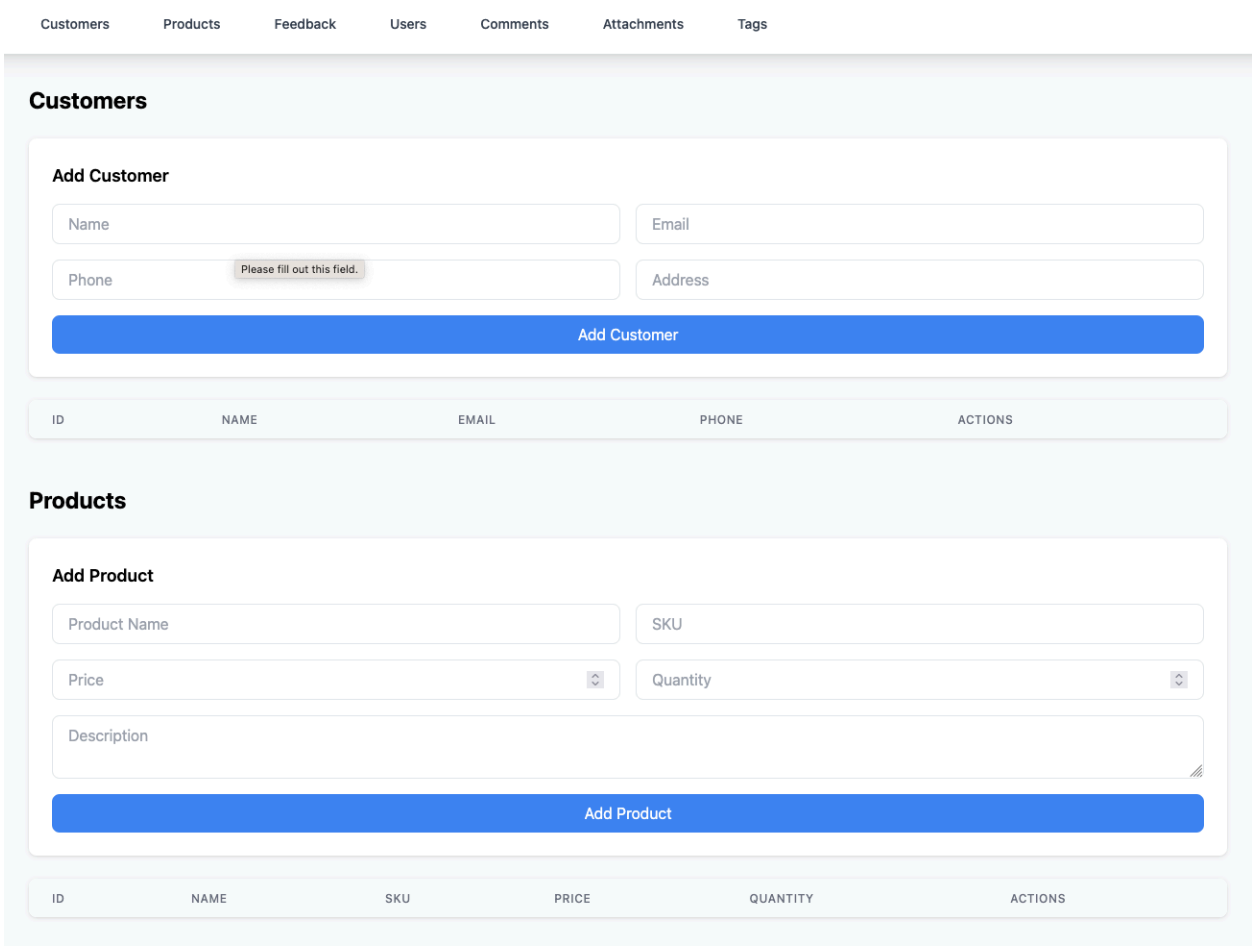
Feedback Assignments

Feedback ID | Assignee ID | Assigned At | Save | Reset

ID	Feedback ID	Assignee ID	Assigned At	Actions

No assignments found.

Anthropic Claude Opus 4-1



Feedback

Add Feedback

Customer ID

Product ID

Select Rating

Subject

Message

Add Feedback

ID	CUSTOMER	PRODUCT	RATING	SUBJECT	ACTIONS

Users

Add User

Username

Email

Password

User

Add User

ID	USERNAME	EMAIL	ROLE	ACTIONS