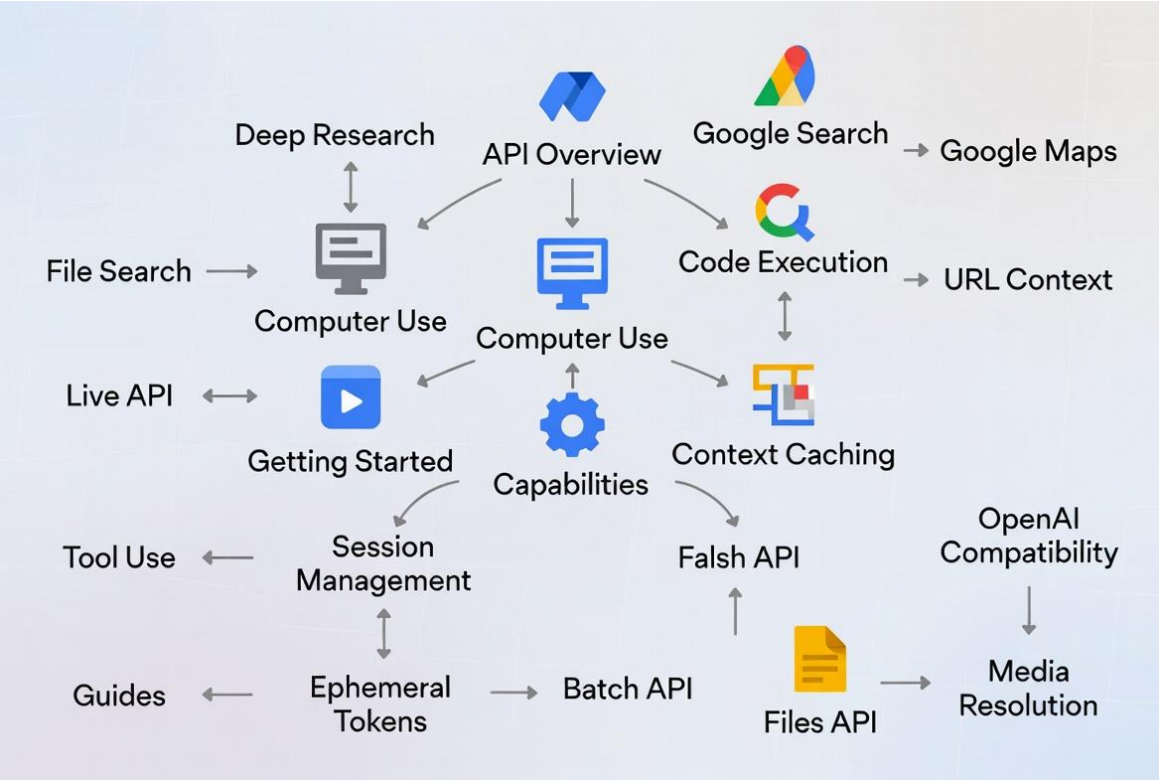
Application Programming Interface



MA. Akmal Fikry

Bachelor of Science Honours in Information Technology

Specialized in Artificial Intelligence

SLIIT.

https://www.kaggle.com/akmal11941

www.linkedin.com/in/akmal-fikry-357b4a321

https://github.com/AkmalFikry027

👉 An API is a bridge that allows two different software applications to communicate with each other.

Instead of directly accessing another program's internal code or database, you **send a request through an API**, and it **returns a response**.

Why Do We Need APIs?

Without APIs:

- Apps would need **direct database access** ❌
- Security would be weak ❌
- Code would be tightly coupled ❌

With APIs:

✅ Secure communication
✅ Platform independence
✅ Reusable services
✅ Scalable systems

Technical Definition (Interview-Level)

An API is a **set of rules, protocols, and tools** that allows one software application to **request and exchange data or functionality** with another application.

Types of APIs

1. **Web APIs (Most Common)**

Used over the internet.

- Google Maps API
- OpenAI API
- Weather API
- Payment APIs (Stripe, PayPal)

2. **Operating System APIs**

- Windows API
- Linux system calls

3. Library / Framework APIs

- Python math module
- Java ArrayList methods
- TensorFlow / PyTorch APIs

Client → API Request → Server → API Response → Client

What is REST API? (Very Important)

REST = Representational State Transfer

4. A **REST API**:

- Uses HTTP methods (GET, POST, PUT, DELETE)
- Uses URLs to identify resources
- Sends data in JSON
- Is stateless

API Security

Common Security Methods:

- API Keys
- OAuth 2.0
- JWT Tokens
- HTTPS

AI/ML, APIs are critical:

- OpenAI API → Chatbots
- Hugging Face API → NLP Models
- TensorFlow Serving → ML Models as APIs

Frontend → API → ML Model → Prediction → API → Frontend

API CREATION IN SPRING BOOT (Backend – Java)

What is a Spring Boot API?

A Spring Boot API is usually a REST API that:

- Accept HTTP requests
- Processes logic
- Returns JSON responses

API Structure (Spring Boot)

- Controller → Service → Repository → Database

HTTP Methods in Spring Boot

Method	Annotation
GET	@GetMapping
POST	@PostMapping
PUT	@PutMapping
DELETE	@DeleteMapping

API CREATION IN **PYTHON**

Flask	FastAPI
Simple	Modern & Fast
Manual validation	Auto validation
Slower	Very fast
Good for basics	Best for AI/ML

API Lifecycle Stages

Design → Develop → Test → Deploy → Maintain → Retire

1️⃣ Design

- Define endpoints
- Choose HTTP methods
- Decide request & response format

2️⃣ Development

- Write controller logic
- Connect to database / model

3️⃣ Testing

- Unit testing
- API testing (Postman)

4️⃣ Deployment

- Cloud / Server
- Docker
- CI/CD

Maintenance

- Bug fixes
- Performance improvement
- Security pitches

6️⃣ Retirement

- Remove unused APIs
- Migrate users to new versions

API Versioning (VERY IMPORTANT)

❓ Why Versioning?

- To avoid breaking existing clients
- To introduce new features safely
- Versioning Methods
- URL Versioning (Most Common)
- Query Parameter Versioning
- Header Versioning

One-Line Exam Definitions

✓ API: A set of rules that allows software applications to communicate
✓ REST API: An API that follows REST principles using HTTP
✓ Endpoint: A specific URL that performs an action
✓ JSON: Lightweight data exchange format
✓ Versioning: Managing multiple API versions

Keep your API key secure

Treat your Gemini API key like a password. If compromised, others can use your project's quota, incur charges (if billing is enabled), and access your private data, such as files.

Critical security rules

- **Keep keys confidential**: API keys for Gemini may access sensitive data your application depends upon.
- **Never commit API keys to source control.** Do not check your API key into version control systems like Git.
- **Never expose API keys on the client-side.** Do not use your API key directly in web or mobile apps in production. Keys in client-side code (including our JavaScript/TypeScript libraries and REST calls) can be extracted.
- **Restrict access**: Restrict API key usage to specific IP addresses, HTTP referrers, or Android/iOS apps where possible.
- **Restrict usage**: Enable only the necessary APIs for each key.
- **Perform regular audits**: Regularly audit your API keys and rotate them periodically.

When building with the **Gemini API**, we recommend using Google **GenAI SDK**. These are the official, production-ready libraries that we develop and maintain for the most popular languages

Tools & Agents with Gemini API

Tools and Agents extend the capabilities of Gemini models, enabling them to take action in the world, access real-time information, and perform complex computational tasks. Models can use tools in both standard request-response interactions and real-time streaming sessions using the Live API.

- **Tools** are specific capabilities (like Google Search or Code Execution) that a model can use to answer queries.

- **Agents** are systems that can plan, execute, and synthesize multi-step tasks to achieve a user goal.

How tools execution work

Tools allow the model to request actions during a conversation. The flow differs depending on whether the tool is built-in (managed by Google) or custom (managed by you).

Built-in tool flow

For built-in tools like Google Search or Code Execution, the entire process happens within one API call:

1. **You** send a prompt: "What is the square root of the latest stock price of GOOG?"
2. **Gemini** decides it needs tools and executes them on Google's servers (e.g., searches for the stock price, then runs Python code to calculate the square root).
3. **Gemini** sends back the final answer grounded in the tool results.

Building agents

Agents are systems that use models and tools to complete multi-step tasks, but they often require an orchestration framework to manage memory, planning, and tool coordination. While Gemini handles reasoning and tool use, reliability improves when prompts explicitly guide planning, persistence, and risk handling. Well-designed system instructions can significantly boost agent performance, improving results on agentic benchmarks by about 5%

Structured outputs vs. function Calling

Gemini offers two methods for generating structured outputs. Use Function calling when the model needs to perform an intermediate step by connecting to your own tools or data systems. Use Structured Outputs when you strictly need the model's final response to adhere to a specific schema, such as for rendering a custom UI.

Agent frameworks

Gemini integrates with leading open-source agent frameworks such as:

- **LangChain / LangGraph**: Build stateful, complex application flows and multi-agent systems using graph structures.

  Ideal for enterprises that need **complex workflows** (e.g., customer support systems, financial analysis pipelines) where multiple AI agents interact and maintain memory across steps.

- **LlamaIndex**: Connect Gemini agents to your private data for RAG-enhanced workflows.

  Connects AI models (like Gemini) to **private, enterprise data** — documents, databases, APIs — enabling **retrieval-augmented generation (RAG)**.

- **CrewAI**: Orchestrate collaborative, role-playing autonomous AI agents.

  Assigns **specialized roles** to AI agents (e.g., analyst, strategist, writer) and orchestrates them to work together on complex tasks.

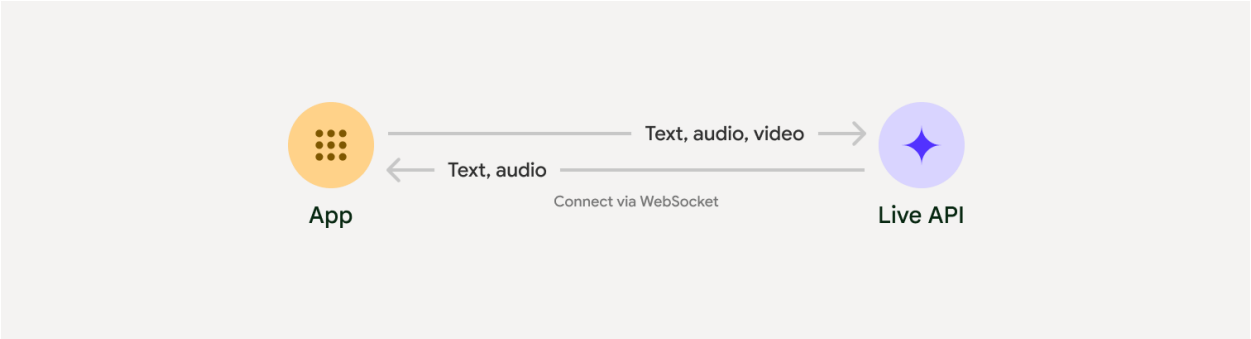
- **Vercel AI SDK**: Build AI-powered user interfaces and agents in JavaScript/TypeScript.

  Let's engineers build **chatbots, assistants, and AI-driven features** directly into web apps using **JavaScript/TypeScript**.

- **Google ADK**: An open-source framework for building and orchestrating interoperable AI agents.

  Standardizes how agents are built and communicate, making it easier to **scale and integrate AI systems across teams and platforms**.

Live API



The Live API enables low-latency, real-time voice and video interactions with Gemini. It processes continuous streams of audio, video, or text to deliver immediate, human-like spoken responses, creating a natural conversational experience for your users

Implementation approaches:

- **Server-to-server**: Your backend connects to the Live API using WebSockets. Typically, your client sends stream data (audio, video, text) to your server, which then forwards it to the Live API.
- **Client-to-server**: Your frontend code connects directly to the Live API using WebSockets to stream data, bypassing your backend.

To streamline the development of real-time audio and video apps, you can use a third-party integration that supports the Gemini Live API over WebRTC or WebSockets.

- Pipecat by Daily

Create a real-time AI chatbot using Gemini Live and Pipecat.

- LiveKit

Use the Gemini Live API with LiveKit Agents.

- Fishjam by Software Mansion

Create live video and audio streaming applications with Fishjam.

- Agent Development Kit (ADK)

Implement the Live API with Agent Development Kit (ADK).

- Vision Agents by Stream

Build real-time voice and video AI applications with Vision Agents.

https://pypi.org/project/PyAudio/(Install helpers for audio streaming. Additional system-level dependencies (e.g. portaudio) might be required. Refer to the link)

Interaction modalities

The following sections provide examples and supporting context for the different input and output modalities available in Live API.

- Sending and receiving audio
- Audio formats
- Audio transcriptions

In addition to the model response, you can also receive transcriptions of both the audio output and the audio input.

To enable transcription of the model's audio output, send *output_audio_transcription* in the setup config. The transcription language is inferred from the model's response.

The Live API supports multiple languages. Native audio output models automatically choose the appropriate language and don't support explicitly setting the language code.

Native audio capabilities

Our latest models feature native audio output, which provides natural, realistic-sounding speech and improved multilingual performance. Native audio also enables advanced features like affective (emotion-aware) dialogue, proactive audio (where the model intelligently decides when to respond to input), and "thinking".

Thinking

The latest native audio output model gemini-2.5-flash-native-audio-preview-12-2025 supports thinking capabilities, with dynamic thinking enabled by default.

The thinkingBudget parameter guides the model on the number of thinking tokens to use when generating a response. You can disable thinking by setting thinkingBudget to 0. For more info on the thinkingBudget configuration details of the model, see the thinking budgets documentation.

Voice Activity Detection (VAD)

Voice Activity Detection (VAD) allows the model to recognize when a person is speaking. This is essential for creating natural conversations, as it allows a user to interrupt the model at any time

Tool use with Live API

Tool use allows Live API to go beyond just conversation by enabling it to perform actions in the real-world and pull in external context while maintaining a real time connection. You can define tools such as Function calling and Google Search with the Live API.

Function calling

Live API supports function calling, just like regular content generation requests. Function calling lets the Live API interact with external data and programs, greatly increasing what your applications can accomplish.

You can define function declarations as part of the session configuration. After receiving tool calls, the client should respond with a list of FunctionResponse objects using the session.send_tool_response method.

See the Function calling tutorial to learn more.

Asynchronous function calling

Function calling executes sequentially by default, meaning execution pauses until the results of each function call are available. This ensures sequential processing, which means you won't be able to continue interacting with the model while the functions are being run.

If you don't want to block the conversation, you can tell the model to run the functions asynchronously. To do so, you first need to add a behavior to the function definitions:

Session management with Live API

A session refers to a persistent connection where input and output are streamed continuously over the same connection (read more about how it works). This unique session design enables low latency and supports unique features, but can also introduce challenges, like session time limits, and early termination.

Session lifetime

Without compression, audio-only sessions are limited to 15 minutes, and audio-video sessions are limited to 2 minutes. Exceeding these limits will terminate the session (and therefore, the connection), but you can use context window compression to extend sessions to an unlimited amount of time.

Context window compression

To enable longer sessions, and avoid abrupt connection termination, you can enable context window compression by setting the contextWindowCompression field as part of the session configuration.

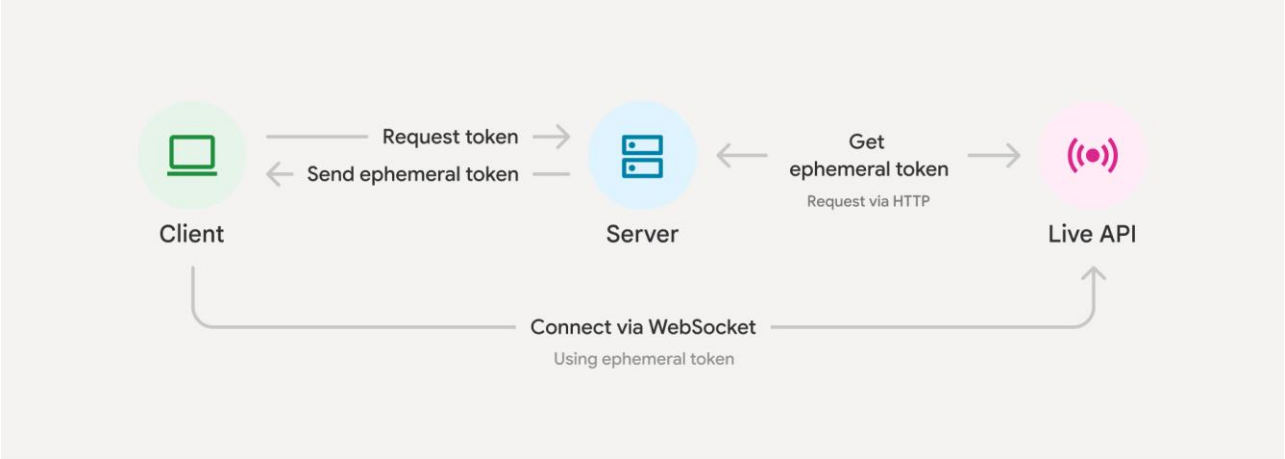
Ephemeral tokens

Ephemeral tokens are short-lived authentication tokens used to securely access the Gemini API via *WebSockets*, especially in client-side applications like web or mobile apps. Although they can still be extracted like regular API keys, their short expiration time and restricted scope greatly reduce security risks. They are recommended for direct client-to-server access to the Live API to improve overall API key security.

How ephemeral tokens work

Here's how ephemeral tokens work at a high level:

1. Your client (e.g. web app) authenticates with your backend.
2. Your backend requests an ephemeral token from Gemini API's provisioning service.
3. Gemini API issues a short-lived token.
4. Your backend sends the token to the client for WebSocket connections to Live API. You can do this by swapping your API key with an ephemeral token.
5. The client then uses the token as if it were an API key.



Reference:

https://ai.google.dev/gemini-api/docs

https://ai.google.dev/gemini-api/docs/models