What technique is used in software apps to determine the type of file being processed (e.g., pdf, rtf, txt, etc). Also how do apps determine what type of file in in a zip file. Is it possible to detect the type at the api level.

Great question—apps rarely rely on the filename alone. Here's how it's actually done and how you can do it at an API boundary.

How file type detection works

1. **Filename extension (fast, weak)**

- Read `.pdf`, `.rtf`, `.txt`, etc.
- Useful hint, but **not trustworthy** (easy to spoof).

2. **MIME type header (hint from caller)**

- From HTTP `Content-Type: application/pdf` or multipart part headers.
- Treat as a hint only; clients often get this wrong.

3. **Magic numbers / file signatures (authoritative, best practice)**

- Inspect the first few bytes of the file ("magic bytes").
- Examples:
  - PDF: `%PDF-` (25 50 44 46 2D)
  - RTF: `{\rtf`
  - PNG: `\x89PNG\r\n\x1A\n`
  - JPEG: `\xFF\xD8\xFF`
  - ZIP: `PK\x03\x04` (and related headers)
  - GZIP: `\x1F\x8B`
- This is what the Unix `file` tool and **libmagic** use.

4. **Deeper/content sniffing (optional)**

- Some formats are containers (e.g., OOXML `.docx`/`.xlsx`/`.pptx` = ZIP).
- Look for internal markers/manifest:
  - **ODF** (e.g., `.odt`): a `mimetype` file at root with the exact MIME type.
  - **OOXML** (Word/Excel/PowerPoint): `[Content_Types].xml` and specific folder structure (`word/`, `xl/`, `ppt/`).
- Tools like **Apache Tika** combine magic bytes + structure analysis.

How to determine file types *inside a ZIP*

- Read the ZIP central directory (no need to extract to disk).
- For each entry:
  - Use the **entry name** as a hint.
  - Optionally **peek into the entry's first bytes** (stream a few bytes) and apply magic-number detection on the uncompressed stream.
- Special cases:
  - If the ZIP itself is an app format:
    - **.docx/.xlsx/.pptx** → identify via `[Content_Types].xml` parts.
    - **.odt/.ods/.odp** → root `mimetype` file (stored uncompressed) gives the MIME type.

Can you detect type at the API level?

Yes—do it server-side. Typical approach for an upload endpoint:

- Accept the upload (`multipart/form-data` or raw body).
- Capture client-provided `Content-Type` (hint).
- **Independently detect** using magic numbers (libmagic/Tika/etc.).
- Enforce an allow-list (e.g., only `application/pdf`, `text/plain`, `application/zip`).
- **Reject mismatches** with `415 Unsupported Media Type`, or down-rank/flag them.
- Optionally log both values for auditing.

Minimal examples

Python (server):

```python
import magic  # pip install python-magic
def detect_mime(file_bytes: bytes) -> str:
    return magic.from_buffer(file_bytes[:4096], mime=True)

# In your upload handler:
# 1) read first ~4–8KB
sniff = uploaded_stream.read(8192)
mime = detect_mime(sniff)
# 2) compare against allow-list and client header
if mime not in {"application/pdf", "text/plain", "application/zip"}:
    raise UnsupportedMediaType(mime)
```

Node.js:

```js
import { fileTypeFromBuffer } from 'file-type'; // npm i file-type

async function detectMime(buf) {
  const ft = await fileTypeFromBuffer(buf);
  return ft?.mime ?? 'application/octet-stream';
}
```

Detect entries inside ZIP (Python):

```
python

import zipfile, io, magic

def inspect_zip(buf: bytes):
    with zipfile.ZipFile(io.BytesIO(buf)) as z:
        for info in z.infolist():
            if info.is_dir():
                continue
            with z.open(info, 'r') as f:
                head = f.read(8192)  # read small chunk
            mime = magic.from_buffer(head, mime=True)
            yield info.filename, mime
```

Platform & library options

- **Cross-platform**: `libmagic` (C) → Python `python-magic`, Node `mmmagic/file-type`, Go `net/http.DetectContentType`, Rust `infer`.
- **"All-in-one" detectors**: **Apache Tika** (Java) parses many formats and containers.
- **OS registries (secondary)**: Windows registry ProgIDs, macOS UTIs; useful for integration, not validation.

Practical recommendations

- **Never trust extensions or client headers alone.**
- **Always sniff bytes** server-side; keep a small read (4–16KB) for performance.
- Validate archives by **inspecting entries**; optionally block risky embedded types.
- Maintain an **allow-list** and return clear errors (`415`).
- For text vs. binary, consider heuristics (printable ratio, BOM detection, UTF-8 validation).
- For security, pair type checks with **AV/malware scanning** and **size/complexity limits** (e.g., max pages for PDFs, max unzip size).

If you share your stack (Python/Node/Java/Go) and framework, I can drop in production-ready middleware for your upload endpoint, plus ZIP entry validation tailored to your accepted formats.