UNIT- IV
CRAWLING APIS AND OCR

Topic: Introduction to APIs

An **API (Application Programming Interface)** is a set of rules that allow different software applications to communicate with each other. APIs define the methods and data formats that programs use to request and exchange information.

In web development, **Web APIs** are commonly used to enable communication between a client (such as a browser) and a server. These APIs use standard HTTP methods like GET, POST, PUT, and DELETE to request data, create new records, update records, or remove data.

APIs are important because they allow developers to access data and functionality from external services, without needing to understand or expose the underlying code.

Example of API use:

- Social media platforms provide APIs that allow developers to fetch user data, post updates, and access analytics.

Topic: Crawling Through APIs

API crawling is a method of collecting data from websites or web applications by interacting directly with their APIs. This is more structured and efficient than traditional web scraping, which involves parsing HTML content.

Advantages of API Crawling:

- **Efficiency**: API crawling is faster and more reliable as it directly interacts with structured data.
- **Less Intrusive**: It respects the website's structure and doesn't break due to layout changes.
- **Data Quality**: APIs typically return clean, structured data in formats like JSON or XML.

API crawling is done using libraries like Python's requests, which helps in sending GET or POST requests to API endpoints and retrieving structured data. Many modern platforms, like Twitter and GitHub, offer public APIs that require authentication and might have rate limits.

Example of API Crawling in Python:

```
import requests

url = 'https://api.example.com/data'
```

```
headers = {'Authorization': 'Bearer YOUR_API_TOKEN'}

response = requests.get(url, headers=headers)

if response.status_code == 200:

    data = response.json()

    print(data)
```

Topic: How APIs Work

APIs work by following a request-response cycle:

1. **Request**: The client sends an HTTP request to the API endpoint with necessary headers and authentication tokens (if required).
2. **Processing**: The server processes the request, accesses data, or performs some operations.
3. **Response**: The server sends back a response, which typically includes a status code (e.g., 200 for success) and data in JSON or XML format.

This process helps in seamless data exchange between different systems.

Topic: Types of APIs

APIs can be categorized into different types based on their usage.

Types of APIs:

1. **Web APIs**:
    - Accessible via HTTP/HTTPS and allow communication between applications over the internet.
    - Examples: REST APIs, SOAP APIs.
    - Common HTTP methods: GET (retrieve data), POST (create data), PUT (update data), DELETE (remove data).
    - Typically return data in JSON or XML format.
2. **Library APIs**:
    - Contain functions or procedures within software libraries.
    - Example: Python's math or os modules.
3. **Operating System APIs**:
    - Interface between the operating system and applications.

- Example: Windows API, POSIX API.

4. **Hardware APIs**:
    - Control hardware devices programmatically.
    - Example: Printer API, GPU APIs.

RESTful APIs are the most common type of Web APIs. They follow the REST principles and use URLs to represent resources. For example:

- GET /users fetches all users.
- POST /users creates a new user.

Here's a simplified version of the data based on the images you uploaded, organized into topics and written in simple language to help students understand and write answers for exams. Below is a breakdown of key topics:

Topic: Use of APIs

APIs (Application Programming Interfaces) are used to automate repetitive tasks and allow different software and platforms to communicate. They help in retrieving real-time data and allow third-party developers to build on existing platforms.

Uses of APIs:

- **Automation**: Helps automate repetitive tasks.
- **Integration**: Allows different software systems to work together.
- **Data Access**: Makes it easy to get real-time, structured data.
- **Extensibility**: Allows third-party developers to build on existing platforms.
- **Scalability**: Helps handle large amounts of data efficiently.

Examples:

- **Social Media**: Twitter API, Facebook Graph API
- **Cloud Services**: AWS API, Google Cloud API
- **Payment Gateways**: PayPal API, Stripe API
- **Maps and Location**: Google Maps API, OpenWeatherMap API

Python’s requests library is often used to interact with APIs. Here's an example of using it to get data from an API:

```
import requests

url = 'https://api.example.com/data'

headers = {'Authorization': 'Bearer YOUR_API_TOKEN'}

response = requests.get(url, headers=headers)

if response.status_code == 200:

    data = response.json()

    print(data)
```

Topic: Key Points on APIs

- APIs are essential for software ecosystems as they allow different systems to communicate and share data.
- They provide access to data or services without revealing the underlying code.
- APIs help developers automate tasks and integrate third-party services, creating a more scalable and efficient software environment.

Topic: Understanding JSON (JavaScript Object Notation)

1. Introduction to JSON

JSON is a lightweight, text-based data format that is commonly used in APIs (Application Programming Interfaces) and web services to exchange data between systems. It is easy for humans to read and write, and it is also easy for machines to parse and generate. JSON represents data in a key-value pair format, making it a simple yet powerful way to handle structured data.

It is used in various programming languages, especially in web development, to exchange data between a server and a client.

2. Structure of JSON

JSON has a very simple and clear structure, consisting of the following main components:

- **Objects**: These are enclosed in curly braces {} and contain key-value pairs. The keys are strings, and each key is associated with a value (which can be a string, number, boolean, array, or another object).

- Example of an object:
- {
- "name": "John",
- "age": 30
- }

- **Arrays**: These are ordered lists of values enclosed in square brackets []. An array can hold multiple values, such as strings, numbers, booleans, or even other arrays or objects.
  - Example of an array:
  - ["apple", "banana", "cherry"]
- **Values**: These can be strings, numbers, booleans (true or false), arrays, or objects.
  - Example values:
    - String: "Hello"
    - Number: 123
    - Boolean: true
    - Array: ["red", "green", "blue"]
    - Object: {"name": "Alice", "city": "New York"}

3. Parsing JSON in Python

Python provides a built-in library called json to handle JSON data. The library helps in converting JSON data into Python objects like dictionaries or lists, and vice versa. There are two main functions used for parsing JSON in Python:

- **json.loads()**: This function is used to parse a JSON string into a Python dictionary or list.
  - Example:
  - import json
  - json_data = '{"name": "DiLKK", "age": 30, "isStudent": false}'
  - python_obj = json.loads(json_data)
  - print(python_obj)

Output:

{'name': 'DiLKK', 'age': 30, 'isStudent': False}

You can access values in the dictionary just like any other Python dictionary:

print(python_obj['name']) # Output: DiLKK

- **json.load()**: This function is used to parse JSON data from a file.
    - Example:
    - import json
    - with open('data.json', 'r') as file:
    - data = json.load(file)
    - print(data)

In this case, data.json contains JSON data that is read and converted into a Python dictionary.

4. Example of JSON Data

Here is a sample JSON data that represents a student's information:

```
{
  "name": "DiLKK",
  "age": 30,
  "isStudent": false,
  "courses": ["Math", "CSE"],
  "address": {
    "city": "Palakollu",
    "zip": "534260"
  }
}
```

- The data has an object that contains several key-value pairs.
- The courses key holds an array of strings (["Math", "CSE"]).
- The address key holds another object with its own key-value pairs.

You can access specific values from this JSON structure using Python:

```
print(data['courses'][0])  # Output: Math

print(data['address']['city'])  # Output: Palakollu
```

5. Conclusion

JSON is an essential data format in modern web development and APIs. Its simplicity and human-readable format make it a great choice for data exchange. In Python, the json library provides a convenient way to work with JSON data, allowing developers to easily parse, modify, and generate JSON objects.

By understanding the structure of JSON and how to parse it, you can efficiently work with data coming from web services or APIs and integrate it into your applications.

Topic: Undocumented APIs

1. Introduction to Undocumented APIs

An **API** (Application Programming Interface) is a set of rules that allows one software or system to interact with another. APIs are used by websites, applications, and services to exchange data. Most APIs are documented, meaning the developers provide guides and information on how to use them. However, there are some **undocumented APIs** that exist but are not officially listed or described in the public documentation.

Undocumented APIs can still be used, but they are often not intended for public use, and their existence may not be openly acknowledged by the service provider. They are typically used in the website's frontend, where JavaScript code fetches data dynamically. Even though these APIs are not officially provided for external use, they may still function and allow access to certain data.

2. Characteristics of Undocumented APIs

- **Not Listed in Documentation**: Undocumented APIs are not publicly shared in any official documentation. This means there is no guide or description of how to interact with these APIs, unlike other standard APIs.
- **Hidden Access**: These APIs provide hidden access to data that may not be easily accessible otherwise. For example, a website might use an undocumented API to retrieve information about

products or users, but this API is not intended for external developers to use.

- **Used by Frontend Code**: The frontend of a website (such as JavaScript running in the browser) may rely on these undocumented APIs to fetch data dynamically. Users can interact with the website, but the backend (server-side) API might not be openly documented.

3. Risks and Concerns of Using Undocumented APIs

- **Unreliable**: Since undocumented APIs are not officially supported, they can break or stop working at any time. The website developers can change or remove these APIs without notice, making them unreliable for long-term use.
- **No Guarantees**: Since there is no official documentation, there is no guarantee that the API will remain available, or that it will work as expected. Relying on undocumented APIs could cause issues in a project if the API becomes unavailable or changes in unexpected ways.
- **Security Risks**: Using undocumented APIs can present security risks, especially if they are not designed with external users in mind. There could be vulnerabilities that expose sensitive data to unauthorized users.

4. Responsible Use of Undocumented APIs

Although undocumented APIs can provide valuable data, they should be used **responsibly**. Here are some guidelines for using them:

- **Avoid Overloading the API**: Since undocumented APIs are not intended for public use, they might not have the same level of performance or scalability as documented APIs. It's important to avoid overloading the API by making too many requests.
- **Monitor for Changes**: Always be aware that the API could change or stop working without notice. It's important to monitor its functionality regularly and have backup plans in case it becomes unavailable.
- **Ethical Considerations**: Be mindful of the website or service's terms of use. Even though an undocumented API might be accessible, using it without permission could violate the terms and conditions of the service.

- **Limited Use**: Undocumented APIs should be used sparingly and only for short-term solutions or specific tasks. If the data is valuable, consider using official, documented APIs whenever possible.

5. Conclusion

In summary, undocumented APIs are endpoints on websites or services that are not officially listed in the documentation. They can provide useful access to data, but they come with risks such as being unreliable and having potential security vulnerabilities. While they can enhance automation and data extraction, developers should use them responsibly and ethically. It is always safer to rely on documented APIs when available.

Topic 1: Web Scraping with APIs

Definition:
Web scraping is the process of extracting data from websites. APIs (Application Programming Interfaces) are commonly used in web scraping because they provide structured data in formats such as JSON or XML.

Key Steps in Using APIs for Web Scraping:

1. **Making API Requests**:
   To access data through APIs, a request is made to a specific API endpoint (URL) using methods like GET, POST, PUT, or DELETE. These requests retrieve structured data from the server. Common headers required in API requests include the Authorization token, User-Agent, and sometimes a Referer header.
2. **Example Request in Python**:
3. import requests
4. headers = {'Authorization': 'Bearer YOUR_TOKEN'}
5. response = requests.get('https://example.com/api/data', headers=headers)
6. if response.ok:
7. data = response.json()
8. print(data)
9. else:

```
print("Request failed with status:", response.status_code)
```

In this example, we make a GET request with headers, check if the response is successful, and print the data.

11. **Handling API Responses**:
    API responses are usually in JSON format, which is parsed using the .json() method in Python. For example:

```
{
   "id": 42,
   "name": "John Doe",
   "email": "john@example.com"
}
```

17. **Common HTTP Status Codes**:
    - **200 OK**: Success, the request was successful.
    - **401 Unauthorized**: Invalid token or access credentials.
    - **404 Not Found**: The requested user or resource does not exist.

Tools for API Usage:

- **Postman**: A tool to organize and test API requests, helpful in exploring APIs.
- **Jupyter Notebooks**: For combining code and documentation.
- **Swagger/OpenAPI**: For formal API documentation.

Topic 2: Finding Undocumented APIs

Definition:
Undocumented APIs are hidden endpoints in a website's backend that are not officially published for public use but are accessible through network requests. These APIs are often used internally by websites to load data dynamically.

Purpose of Undocumented APIs:

1. **Access Dynamic Content**:
   These APIs are often used to fetch live data from websites, which is not visible in the standard HTML content.

2. **Automate Data Retrieval**:
   When there is no public API, undocumented APIs allow developers to automate the process of fetching data.
3. **Understanding Website Operations**:
   They help understand how a website operates behind the scenes by inspecting the network activity.

Tools to Discover Undocumented APIs:

- **Browser Developer Tools (DevTools)**: Inspect network requests to discover hidden endpoints.
- **Postman**: Used to test APIs.
- **Fiddler/Charles Proxy**: Monitors HTTP(S) traffic.
- **Burp Suite**: Advanced tool for request inspection and security testing.

Step-by-Step Process to Discover Undocumented APIs Using DevTools:

1. **Open DevTools**: Right-click on a webpage and select "Inspect," then go to the "Network" tab.
2. **Perform an Action**: Click buttons or perform searches to load dynamic content on the website.
3. **Filter Requests**: Look for requests related to data fetching, such as XHR or Fetch requests.
4. **Examine a Request**: Click on a request to view its details, such as URL, method, headers, and response data.

Advanced Techniques:

- **JavaScript Files**: Check for .js files in the "Sources" tab that may contain API calls.
- **Pagination**: Look for URL parameters like ?page=2 to fetch more data.
- **Request Patterns**: Observe the headers, such as CSRF tokens or session cookies, which may be required for making requests.

Topic 3: Documenting APIs

Definition:
Documenting APIs refers to the process of creating reference

documentation that explains the structure and usage of APIs. It is essential for understanding and using APIs effectively.

Importance of Documenting APIs:

1. **Share Knowledge**: API documentation helps teams understand and utilize APIs.
2. **Track Changes**: It keeps track of changes in API endpoints and parameters over time.
3. **Simplify Maintenance**: It helps maintain and update scripts or tools that depend on APIs.

Essential Elements of API Documentation:

- **Endpoint URLs**: The full URL or base path of the API.
- **HTTP Methods**: The request methods such as GET, POST, PUT, DELETE.
- **Headers**: Authentication (API keys, bearer tokens), Content-Type, User-Agent, etc.
- **Parameters**: URL query parameters or body fields.
- **Request Examples**: Sample requests, either in cURL or Python format.
- **Response Format**: Example response structure, usually in JSON or XML format.
- **Status Codes**: Documenting standard HTTP status codes like 200, 404, and 401.

Training Tesseract (OCR)

Tesseract: Open-source OCR tool to convert images → editable text.

Why Training? To handle special tasks (CAPTCHAs, new fonts, handwriting).

Steps to Train:

1. Collect Image Samples – Gather text images (fonts, styles).

2. Label Data – Provide correct text for each image.

3. Preprocessing – Resize, grayscale, noise removal.

4. Training Process – Use Tesseract training tools to learn characters.

5. Testing – Run OCR on new images and improve accuracy.

Reading CAPTCHAs

CAPTCHA: Completely Automated Public Turing test to tell Computers and Humans Apart.

Purpose: Block bots on websites.

Types:

1. Text CAPTCHAs – Distorted letters/numbers.

2. Image CAPTCHAs – Identify objects (cars, traffic lights).

3. Audio CAPTCHAs – Spoken words/numbers for accessibility.

Challenges: Distortion, background noise.

Solution: OCR (Tesseract) + preprocessing (grayscale, noise removal) + Deep Learning.

Finding and Documenting APIs Automatically

API: Allows software communication.

Why Needed? Saves developer time, avoids errors.

Steps:

1. Discovery – Tools scan and find APIs.

2. Analysis – Study endpoints, requests, and responses.

3. Documentation – Export as OpenAPI/JSON/HTML.

Tools: Swagger, Postman.

Benefits: Fast, error-free, helps integration.

Processing Well-Formatted Text

Formats: PDF, HTML, Markdown, etc.

Libraries: Pillow, PyPDF2, BeautifulSoup.

Steps:

1. Read file → 2. Convert/Preprocess → 3. Extract useful text → 4. Store/Analyze.

Applications: Reports, notes digitization, text mining.

Image Processing & Text Recognition

Image Processing: Manipulating images to improve/ analyze.

Text Recognition (OCR): Extracting text from images.

Stages:

1. Preprocessing – Grayscale, noise removal, thresholding.

2. OCR – Use Tesseract.

3. Postprocessing – Correct text, format neatly.

Applications: Books → e-text, license plates, passports.

Libraries for OCR & Image Processing

OpenCV: Image operations (resize, blur, threshold).

PIL/Pillow: Read/edit images in Python.

Tesseract: OCR text extraction.

Uses: Automate text reading from scanned files, screenshots, forms.

Types of CAPTCHAs

1. Text CAPTCHA – Distorted text.

2. Image CAPTCHA – Select objects.

3. Audio CAPTCHA – Listen & type.

Use: Prevent bots.

Challenge: Hard for OCR, easy for humans.

UNIT- IV
CRAWLING APIS AND OCR

Summary Table

Topic	Key Points	Tools/Uses
Training Tesseract	Collect → Label → Train → Test	Fonts, Handwriting OCR
Reading CAPTCHAs	Text, Image, Audio	Security, Bot prevention
API Discovery	Scan → Analyze → Document	Swagger, Postman
Well-Formatted Text	Extract from PDF/HTML/Markdown	Reports, Digitization
Image Processing & OCR	Preprocess → OCR → Postprocess	Books, Passport, License plates
Libraries	OpenCV, Pillow, Tesseract	Image editing + OCR

Image Processing and Text Recognition

Definition

Image Processing is a technique in computer science used to enhance, analyze, or extract useful information from digital images.

Text Recognition (OCR) is the process of detecting and converting text in images into editable and searchable form.

Key Concepts

Optical Character Recognition (OCR): Converts text from images/scanned files into editable text.

Libraries Used:

OpenCV → image preprocessing (resize, grayscale, filtering).

Tesseract → text recognition.

Applications

Converting printed/scanned books into digital text.

Passport/ID verification at airports.

Automatic Number Plate Recognition (ANPR).

Extracting text from screenshots and PDFs.

Steps in OCR Workflow

1. Image Reading – Load the image using libraries.
2. Preprocessing – Resize, grayscale, thresholding to improve clarity.
3. Text Extraction – Apply OCR (e.g., Tesseract).
4. Postprocessing – Correct and format recognized text.

Extracting Text from PDFs & Screenshots

Screenshots/PDFs are image-based → require OCR.

Preprocessing (grayscale, contrast enhancement) increases accuracy.

Workflow: Capture → Preprocess → Apply OCR → Save text.

Automatic API Discovery & Documentation

Definition

APIs (Application Programming Interfaces) allow software to communicate.

Automatic discovery helps in identifying available APIs and generating documentation.

Tools

Swagger – Generates interactive API documentation.

Postman – Tests APIs and creates docs automatically.

Steps

1. Discovery – Scan application and detect APIs.
2. Analysis – Check inputs (requests) and outputs (responses).

3. Documentation – Export in user-friendly formats (HTML/JSON).

Benefits

Saves developer time.

Reduces errors in API usage.

Improves software integration.

Summary Table

Topic	Key Points	Applications
Image Processing & OCR	Image reading, preprocessing, OCR	Books, Passports, License plates
Text from PDFs	OCR + preprocessing	Notes digitization, Text analysis
API Discovery	Automated tools, Swagger/Postman docs	Developer productivity, Integration

---

CAPTCHAs and OCR

Introduction to CAPTCHAs

CAPTCHA → Completely Automated Public Turing test to tell Computers and Humans Apart.

Purpose → To check if a user is a human, not a bot.

Example → While creating a Gmail account, you are asked to type distorted letters.

Types of CAPTCHAs (with examples)

1. Text-based CAPTCHA → Distorted letters/numbers.

Example: "Ty9WQ" shown with curves and lines.

2. Image-based CAPTCHA → Identify objects in images.

Example: "Select all squares with traffic lights."

3. Audio CAPTCHA → User listens to numbers/words.

Example: "Four, Eight, Nine" played with background noise.

Working of CAPTCHAs

CAPTCHAs are easy for humans but difficult for machines.

Example: Humans can still read distorted letters, but OCR software gets confused due to noise, overlapping lines, and random colors.

Challenges in Breaking CAPTCHAs

Extra noise and background patterns.

Distorted shapes and random fonts.

Overlapping letters.

Example: Word "EXAM" written in zig-zag with dots and lines → hard for OCR.

Techniques to Solve/Recognize CAPTCHAs

1. Preprocessing → Convert to grayscale, remove noise.

2. Segmentation → Split into individual letters.

3. OCR Engines → Use tools like Tesseract.

4. Deep Learning → CNN models for high accuracy.

Example: A CNN model trained on thousands of CAPTCHA samples can solve distorted text better.

OCR – Optical Character Recognition

Definition → Converts images/scanned documents → editable text.

How it works → Analyzes shapes of characters in the image.

Example: Scanning a printed page of a textbook and converting it into a Word file.

UNIT- IV

CRAWLING APIS AND OCR

Processing Well-Formatted Text with OCR

OCR supports file formats like PDF, JPEG, PNG, TIFF.

Can handle tables, forms, structured and unstructured documents.

Example: A company scans paper bills and OCR extracts amounts and dates into Excel.

Applications (CAPTCHAs + OCR)

Web Security → Stop bots from fake registrations.

Digital Libraries → Scanning old books.

Accessibility → Text-to-speech for visually impaired.

Business Automation → Data entry from invoices.

Example: Indian Railways uses OCR to scan passenger forms quickly.

Limitations

Very complex CAPTCHAs may confuse even humans.

OCR may fail on poor quality/blurred images.

Advanced bots with AI can sometimes solve old CAPTCHA types.

Quick Summary Table

Topic	Key Points	Example
CAPTCHA	Human vs bot test	Gmail signup check
Types	Text, Image, Audio	Traffic lights selection
Working	Easy for humans, hard for OCR	Distorted letters with noise
Techniques	Preprocess, Segment, OCR, CNN	Tesseract, Deep Learning
OCR	Converts images → text	Scanned book → Word file
Applications	Security, Digital archive, Data	Indian Railways forms
Limitations	Hard CAPTCHAs, poor image errors	Blurry scanned page fails OCR