Session 2: Working with API Data and API Documentation 📜

Delve deeper into handling and manipulating data obtained from APIs, understanding different data formats, and exploring how to use and read API documentation effectively.

📽 Link to Presentation

Slide 1: Title

Slide 2: Agenda

🧑‍🏫 Session 1 Review

▶ Click to Expand

Slide 4: Previous Topics

A mini-agenda to review topics from the previous session

- What are APIs?
- API Architecture
- HTTP

❓ What does API stand for?

Slide 6: Definition of APIs

Overview of what an API is with examples

> An Application Programming Interface is a set of protocols that allows different software applications to communicate, interact, and share data with each other.

- Watch video for good, simple explanation
- Additional examples of APIs
  - **Weather Apps**: Weather apps use APIs to access real-time weather data from external sources. These APIs provide accurate and up-to-date information. By leveraging APIs, weather apps avoid the need to collect and maintain their own weather data.
  - **Social Media**: When you click "Share", an API is invoked, sending the data to the respective social media platform. The platform's API processes the request, posts the content, and provides feedback to the user.
  - **Payment Apps**: When you initiate a payment, the app sends transaction details to the payment gateway's API. The API handles payment authorization, processes the transaction, and returns a response to the app

❓ What does API stand for?

Slide 8: Case Studies

McBroken

The McBroken app uses the McDonald's API to track the availability of working ice cream machines at various locations in real-time, providing users with up-to-date information on whether they can get frozen treats.

- Software Developer reverse-engineered the McDonald's ordering API to send an order worth $18,752 of McFlurries to every McDonald's in the US
- Based on whether the item can be added to your cart determines if the machine is working or not

Pokemon Go

Pokémon Go is an augmented reality mobile game that uses real-world locations and the camera on players' smartphones to allow them to catch virtual Pokémon in their surroundings.

- Utilizes the Google Maps API to display Pokemon in your environment

Procore Permissioning

Procore is a cloud-based construction management platform that provides tools for project management, collaboration, scheduling, and financial management.

- Procore provides permissions templates that sometimes can only be applied on a per-person basis meaning that.
- If we wanted to specify *everyone*'s permissions for a given project, someone would have to go through each individual and update their permissions.
- We can use the Procore API to do this for us by automating the process. We still have to go one-by-one, but the computer can change someone's permissions in a matter of milliseconds while it might take a user 10 seconds to do the same process (not to mention it would be incredibly boring).

❓ What are the two main types of API architecture?

Slide 10: API Architecture

How the rules of an API are setup to ensure smooth communication

SOAP (Simple Object Access Protocol):

- Like sending a package with instructions and details
- More structured and formal than REST
- Often used in big businesses
- Communication is more like writing a letter: you need to follow specific rules
- Can use different delivery methods (transport protocols) like HTTP, SMTP (email), etc.
- Has a fixed structure (XML) for messages, making sure everyone understands the message format

REST (Representational State Transfer):

- Modern and simple way for software to communicate over the internet
- Communication is like talking to a waiter: you ask for things (GET), give new things (POST), update (PUT), or remove (DELETE)
- Simple and straightforward

- Uses URLs to represent different resources (like menu items), and you use different actions (HTTP methods) to interact with those resources

Slide 11: Process Overview

How the API process actually works

1. Client (that is you) makes a request using HTTP
2. Server (that is the program you are accessing) processes the requests, performing the action that you specify (if it can)
3. Response is generated in HTTP and sent back to you, the Client
4. Client processes the response

Slide 12: HTTP Structure

Start Line

First line of the request/response

For requests, the start line is called the "Request Line" and includes:

- HTTP method
- URL of the resource being requested
- Parameters
- version of the HTTP protocol being used

For responses, the start line is called the "Status Line" and includes:

- three-digit status code
- text description of status
- version of the HTTP protocol being used

Headers

Additional lines that include important, standardized information for the HTTP request/response

You can find available Header options here, but some of the more common ones include:

- **Authorization**: credentials
- **Content-Type**: media type for the body of the request/response
- **Host**: domain name of the server i.e. google.com

Blank Line

Tells the program that the previous values were for the header while the following are for the body

Body

Optional component that carries additional data sent with the request/response, such as form data or request payload.

For requests, the body is often formatted in:

- JSON
- XML

For responses, the body is often formatted in:

- JSON
- XML
- HTML

❓ What are some of the common HTTP methods?

Slide 14: HTTP Methods

The data manipulation methods used by APIs

There are 9 HTTP methods in HTTP v1.1, but there are four/five ones that are commonly used:

- **POST**: Used to send data to the server, for example, customer information, file upload, etc.
- **GET**: Used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
- **PUT**: Replaces all current representations of the target resource with the uploaded content.
- **PATCH**: Applies partial modifications to a resource.
- **DELETE**: Removes the specified resource.

Slide 15: POST Request

```
POST /api/users HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "username": "newuser",
    "email": "newuser@example.com",
    "password": "securepassword"
}
```

Identify the key components:

1. Request Line
    - Method: `POST`
    - URL: `/api/users`
    - HTTP Version: `HTTP/1.1`
2. Headers
    - Header 1: `Host: example.com`
    - Header 2: `Content-Type: application/json`
3. Blank Line
4. Body

  - JSON Form:

    ```
    {
        "username": "newuser",
        "email": "newuser@example.com",
        "password": "securepassword"
    }
    ```

Slide 16: POST Response

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": 123,
    "username": "newuser",
    "email": "newuser@example.com"
}
```

Identify the key components:

1. Status Line
   - HTTP Version: `HTTP/1.1`
   - Status Code: `201`
   - Status Text: `Created`
2. Headers
   - Header 1: `Content-Type: application/json`
3. Blank Line
4. Body
   - JSON Form:

    ```
    {
        "id": 123,
        "username": "newuser",
        "email": "newuser@example.com"
    }
    ```

Slide 17: Response Status Codes

Status of the HTTP request

100s - Informational

An informational response indicates that the request was received and understood. It is issued on a provisional basis while request processing continues. It alerts the client to wait for a final response.

200s - Success

These status codes indicates the action requested by the client was received, understood, and accepted. Common success status codes include (but are not limited to):

- **200 OK**: Standard response for successful HTTP requests. The actual response will depend on the request method used.
- **201 Created**: The request has been fulfilled, resulting in the creation of a new resource.

300s - Additional Steps

This class of status code indicates the client must take additional action to complete the request. Many of these status codes are used in URL redirection.

400s - Client-side Error

This class of status code is intended for situations in which the error seems to have been caused by the client. Some common 400 status codes are:

- **400 Bad Request**: The server cannot or will not process the request due to an apparent client error (bad request syntax, size too large, invalid request message framing, or deceptive request routing)
- **401 Unauthorized**: For use when authentication is required and has failed or has not yet been provided
- **403 Forbidden**: The request contained valid data and was understood by the server, but the server is refusing action. This may be due to the user not having the necessary permissions for a resource or needing an account of some sort, or attempting a prohibited action.
- **404 Not Found**: The requested resource could not be found but may be available in the future. Subsequent requests by the client are permissible.

500s - Server-side Error

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has encountered an error or is otherwise incapable of performing the request.

ℹ API Data Formats and Handling Responses

▶ Click to Expand

❓ What are the two msot common API data formats?

Slide 20: XML

What is XML and what are some key characteristics

XML (eXtensible Markup Language) is a widely used format for structuring and representing data that needs to be exchanged between different software systems. XML is not limited to APIs; it's also used for data storage, configuration files, and more.

- **Definition**: XML is a markup language that defines rules for encoding documents in a format that is both human-readable and machine-readable.
- **Purpose**: XML is used to structure and represent data in a hierarchical and standardized way, making it easier for different software applications to understand and process the data.

XML in API Context:

- **Data Exchange**: APIs often use XML as a format for exchanging data between a client (requester) and a server (provider).
- **Data Representation**: XML allows you to represent structured data with elements, attributes, and values, making it suitable for various types of information like configuration settings, lists, documents, and more.
- **Data Types**: XML supports a wide range of data types, including strings, numbers, dates, and more complex structures.
- **Flexibility**: XML is extensible, meaning you can define custom tags and structures to represent specific data formats or schemas.
- **Human-Readable**: XML documents are designed to be both human-readable and machine-readable. This readability helps developers understand the structure and content of the data being exchanged.

XML Structure:

- **Tags**: XML documents use tags to define elements that hold data. Tags are enclosed in angle brackets, like `<tag>`.
- **Attributes**: Elements can have attributes that provide additional information. Attributes are defined within the opening tag, like `<element attribute="value">`.
- **Hierarchy**: XML documents have a hierarchical structure, with elements nested within other elements to create a tree-like arrangement.
- **Closing Tags**: Each opening tag must have a corresponding closing tag (e.g., `<tag>data</tag>`).

Slide 21: XML Example

```
<bookstore>
    <book category="Fiction">
        <title>The Great Gatsby</title>
        <author>F. Scott Fitzgerald</author>
        <price>10.99</price>
    </book>
    <book category="Non-Fiction">
        <title>Sapiens</title>
        <author>Yuval Noah Harari</author>
        <price>15.95</price>
    </book>
</bookstore>
```

Slide 22: JSON Data

What is JSON and what are some key characteristics

JSON (JavaScript Object Notation) is:

- **Definition**: a lightweight data interchange format that is easy for both humans and machines to read and write.
- **Purpose**: used to structure and represent data in a format that is efficient for data exchange between different software systems.

JSON in API Context

- **Data Exchange**: APIs often use JSON as a format for exchanging structured data between clients and servers.
- **Data Representation**: JSON represents data as key-value pairs, arrays, and nested objects, making it suitable for various types of information.
- **Data Types**: JSON supports basic data types such as strings, numbers, booleans, arrays, and objects.
- **Simplicity**: JSON's syntax is less verbose than XML, which contributes to its simplicity and ease of use.

JSON Structure

- **Objects**: JSON data is organized into objects, which consist of key-value pairs enclosed in curly braces ({}).
- **Arrays**: Arrays in JSON are ordered lists of values enclosed in square brackets ([]).
- **Values**: Values can be strings, numbers, booleans, objects, arrays, or null.
- **Keys**: Keys are strings that represent the names of values within objects.

Slide 23: JSON Example

```
{
    "bookstore": {
        "books": [
            {
                "category": "Fiction",
                "title": "The Great Gatsby",
                "author": "F. Scott Fitzgerald",
                "price": 10.99
            },
            {
                "category": "Non-Fiction",
                "title": "Sapiens",
                "author": "Yuval Noah Harari",
                "price": 15.95
            }
        ]
    }
}
```

🔑 Key Points (Slide 24)

1. **XML (eXtensible Markup Language)**:

    - XML is a human-readable and machine-readable markup language that structures data in a standardized way, commonly used in APIs.
    - XML offers flexibility through custom tags and structures, with data organized using tags, attributes, and a hierarchical tree structure.
2. JSON (JavaScript Object Notation)
    - JSON is a lightweight data interchange format that is efficient for data exchange, especially between different software systems.
    - JSON represents data using key-value pairs and has a simpler syntax compared to XML, making it easier to use.

🔐 API Access and Security

▶ Click to Expand

Slide 26: Authentication versus Authorization

Differences between the widely interchanged words

Authentication

Authentication is the process of verifying the identity of a user, system, or entity. It ensures that the person or entity claiming access to a system or resource is indeed who they say they are. Authentication is the first step in the security process and is typically based on providing credentials such as a username and password, a security token, a fingerprint, or other forms of identity verification. Once authenticated, a user gains access to a system or application.

Authorization

Authorization, on the other hand, comes after authentication and involves granting or denying access to specific resources or actions based on the authenticated user's permissions. In essence, authorization determines what actions a user or entity is allowed to perform within a system or application. Authorization is typically defined by roles, permissions, or access levels that are associated with the authenticated user. It ensures that users only have access to the functionalities and data they are entitled to based on their roles or privileges.

TL;DR

- **Authentication** is about confirming the identity of a user.
- **Authorization** is about granting or denying access to specific resources or actions based on the user's verified identity and permissions

Slide 27: Common Authentication Methods

API Keys

- API keys are simple and widely used for authentication.
- They are unique alphanumeric strings issued to clients (applications or users) by the API provider.
- Clients include the API key in the request headers or query parameters to authenticate themselves.

- API keys are suitable for public APIs with lower security requirements.

Bearer Token Authentication (Token-based)

- Bearer token authentication is used with tokens like JWT (JSON Web Token) or OAuth 2.0 access tokens.
- After successful authentication, clients receive an access token, which they include in the request headers.
- The server validates the token to authorize the client's access to resources.
- Bearer token authentication provides flexibility and scalability.

Basic Authentication

- Basic Authentication involves sending a username and password in the request headers.
- The credentials are typically base64-encoded (but not encrypted), making it important to use HTTPS for secure transmission.
- While simple to implement, Basic Authentication is less secure due to the risk of credentials being intercepted.

Slide 28: Basic Authentication

Standard username and password

A developer accessing a private GitHub repository using the Git command-line tool. The tool prompts for a username and password, which the developer provides. The credentials are then base64-encoded and included in the Git request headers for authentication.

```
git clone https://github.com/username/repo.git
Username: your_username
Password: your_password
```

Slide 29: API Keys

Keys provided in header or query parameters

Using a weather API to fetch weather information for your application. The API provider gives you a unique API key. To authenticate, you include the API key in the request URL when making API calls.

```
GET https://api.weather.com/forecast?api_key=your_api_key
```

Slide 30: Token-based

Similar to keys but token expires

A mobile app that interacts with a user's social media account. After the user logs in, the app receives a JWT (JSON Web Token). When making requests to the social media API, the app includes the JWT in the request headers.

```
GET https://api.socialmedia.com/posts
Authorization: Bearer your_jwt
```

🔍 Key Points (Slide 31)

Summary of the API Access and Security section

1. **Authentication vs. Authorization**: Authentication is the process of verifying a user's identity (e.g., through username/password), while authorization determines the access or actions a user can perform based on their permissions.
2. **API Keys**: Simple alphanumeric strings given by API providers to authenticate clients. Often used for public APIs with lower security requirements and can be passed in headers or query parameters.
3. **Bearer Token Authentication**: Uses tokens, like JWT or OAuth 2.0 access tokens. Upon successful authentication, clients receive an access token to include in request headers, offering flexibility and scalability.
4. **Basic Authentication**: Involves sending base64-encoded usernames and passwords in request headers. It's simple but less secure due to potential interception risks.

📝 API Documentation

▶ Click to Expand

Slide 33: Navigating Docs

A guidebook to the API's capabilities and usage

Key components of API documentation include:

- **Endpoints**: These are URLs that define specific functions or resources the API provides.
- **Parameters**: These are inputs required to customize your API requests, such as query parameters, headers, or request bodies.
- **Responses**: Documentation explains what data the API returns in response to different requests.
- **Authentication**: Details about how to authenticate and authorize your requests using API keys, tokens, or other methods.

Some examples:

- Procore API Docs
- OpenWeatherMap API Docs
- NASA API Docs

Slide 34: Endpoints

More details on endpoints

- **Definition**: Endpoints are specific URLs that represent different functions or resources within an API. Each endpoint corresponds to a particular action or retrieval of data.

- **Purpose**: Endpoints act as the entry points for clients (applications or users) to interact with an API. They provide a structured way to access specific functionalities offered by the API.
- **Usage**: Clients use different HTTP methods (GET, POST, PUT, DELETE, etc.) on specific endpoints to perform actions like retrieving data, creating new records, updating existing records, or deleting data.

GET Endpoint for Retrieving User Information:

- **Endpoint**: /api/users/{user_id}
- **Description**: This endpoint is used to retrieve information about a specific user identified by their user_id.
- **HTTP Method**: GET ** Example Request**:

```
GET /api/users/123
```

POST Endpoint for Creating a New Post:

- **Endpoint**: /api/posts
- **Description**: This endpoint allows clients to create a new post.
- **HTTP Method**: POST
- **Example Request**:

```
POST /api/posts
{
  "title": "New Post Title",
  "content": "This is the content of the new post."
}
```

Slide 35: Parameters

The three types of paremeters and what their usage is

Header Parameters

- **Purpose**: Header parameters contain additional information about the request or the client making the request.
- **Usage**: Header parameters are included in the headers section of the HTTP request. They provide context or instructions for the server to process the request properly. Examples include authentication tokens, user agents, and content types (e.g., JSON or XML).

Path Parameters

- **Purpose**: Path parameters allow dynamic segments in the URL path to identify specific resources or actions.
- **Usage**: Path parameters are inserted directly into the URL path and enclosed within curly braces. They are used to specify identifiers, such as IDs or slugs, that help the server determine which resource the

client is requesting. For instance, in a URL like `/users/{user_id}`, `user_id` is a path parameter.

Query Parameters

- **Purpose**: Query parameters enable customization and filtering of API requests by providing additional information to the server.
- **Usage**: Query parameters are appended to the URL after a question mark (?). They are in the form of key-value pairs, separated by &. Query parameters help modify the behavior of the request, such as specifying search terms, filters, sorting options, or pagination limits. For example, in a URL like `/products?category=electronics&sort=price`, `category` and `sort` are query parameters.

Slide 36: Parameters Example

How parameters are documented

Documentation for Path Parameters typically includes:

- **Names**: Documentation lists the names of path parameters that you may need to include in your API request's URL.
- **Types**: The data types or formats expected for each path parameter. This information helps you ensure that your parameter values match the expected type.
- **Placement**: Path parameters are typically included directly in the URL's path, and the documentation shows where in the URL to place each parameter.
- **Constraints**: Documentation may outline any constraints or validation rules that path parameters must adhere to, such as minimum/maximum lengths or allowed characters.
- **Required Parameters**: Whether a path parameter is required or optional. If a parameter is required, it must be included in the URL for the request to be valid.
- **Example URLs**: Documentation often provides examples of complete URLs with path parameters included. These examples serve as templates for constructing your requests.
- **Descriptions**: You'll find descriptions or explanations for each path parameter, helping you understand its purpose and usage within the API request.

Documentation for Query Parameters typically includes:

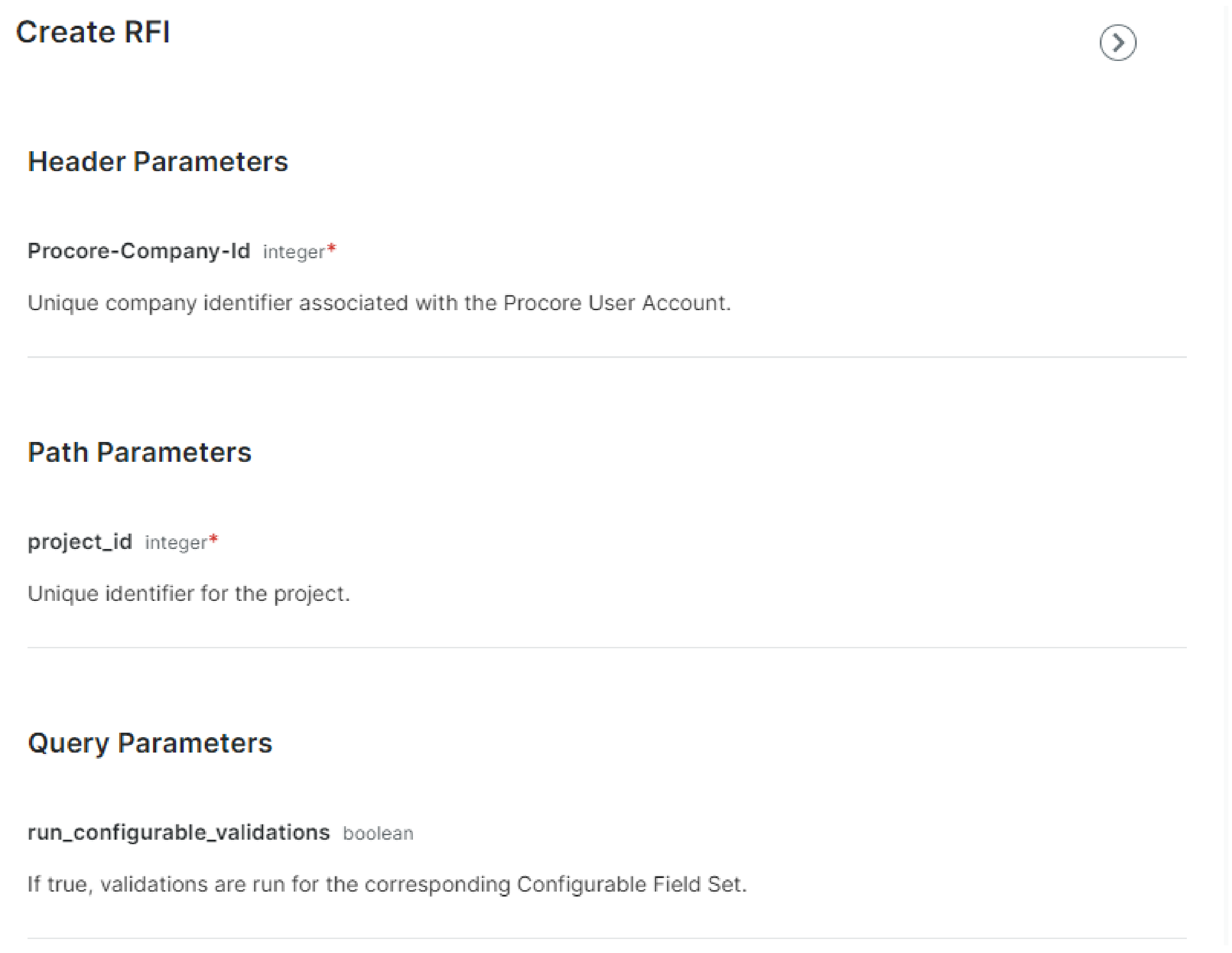
- **Names**: Documentation lists the names of query parameters that you may include in your API request's URL. These parameters are typically added after the "?" character in the URL.
- **Types**: It specifies the data types or formats expected for each query parameter, helping you ensure that your parameter values match the expected type.
- **Usage**: Query parameters are typically used to filter, paginate, or customize API responses. Documentation explains how each parameter affects the response.
- **Constraints**: Documentation may outline any constraints or validation rules that query parameters must adhere to, such as valid values or numeric ranges.
- **Optional Parameters**: Query parameters are often optional, meaning you can choose whether to include them in the URL. Documentation indicates which parameters are optional and which are required.
- **Example URLs**: Documentation provides examples of complete URLs with query parameters included, showing how to structure requests for specific use cases.

- **Parameter Descriptions**: You'll find descriptions or explanations for each query parameter, helping you understand its purpose and how it influences the API response.

Documentation for Headers typically includes:

- **Names**: Documentation lists the names of the headers that you may need to include in your API request. These names are case-sensitive and should be entered exactly as specified.
- **Values**: For each header, it specifies the expected or allowed values. Some headers may have specific values that are required or optional.
- **Examples**: Documentation often provides examples of header configurations for different scenarios.
- **Header Descriptions**: You'll find descriptions or explanations for each header, clarifying its purpose and how it affects the API request or response.
- **Required Headers**: It specifies whether a header is required or optional. If a header is required, it's crucial to include it in your request; otherwise, the request may fail.
- **Format**: Information about the format or syntax of headers is given. For instance, some headers require a specific format, like date and time in a specific format.
- **Authentication Headers**: If the API uses authentication, the documentation provides details about headers required for authentication, such as API keys or tokens.

Below is the Procore API Documentation on how to create a RFI using a POST request:



Create RFI

Header Parameters

Procore-Company-Id integer*

Unique company identifier associated with the Procore User Account.

Path Parameters

project_id integer*

Unique identifier for the project.

Query Parameters

run_configurable_validations boolean

If true, validations are run for the corresponding Configurable Field Set.

You can see we have:

- **Header Parameter**: `Procore-Company-Id` which needs to be an integer
- **Path Parameter**: `project_id` which also needs to be an integer
- **Query Parameter**: `run_configurable_validations` which is a boolean

The actual request in HTTP might look something like this:

```
POST /rest/v1.0/projects/681425/rfis?run_configurable_validations=False HTTP/1.1
Host: api.procore.com
Authorization: Bearer REPLACE_BEARER_TOKEN
Procore-Company-Id: 8089
Content-Type: application/json

{
  "rfi": {
    "subject": "Wall Color",
    "reference": "Color of the kitchen wall"
  }
}
```

Where:

- **Header Parameters**: All parameters under the URL such as `Authorization` and `Content-Type`
- **Path Parameter**: `681425` is placed in the URL
- **Query Parameter**: `run_configurable_validation` is placed after the question mark "?"

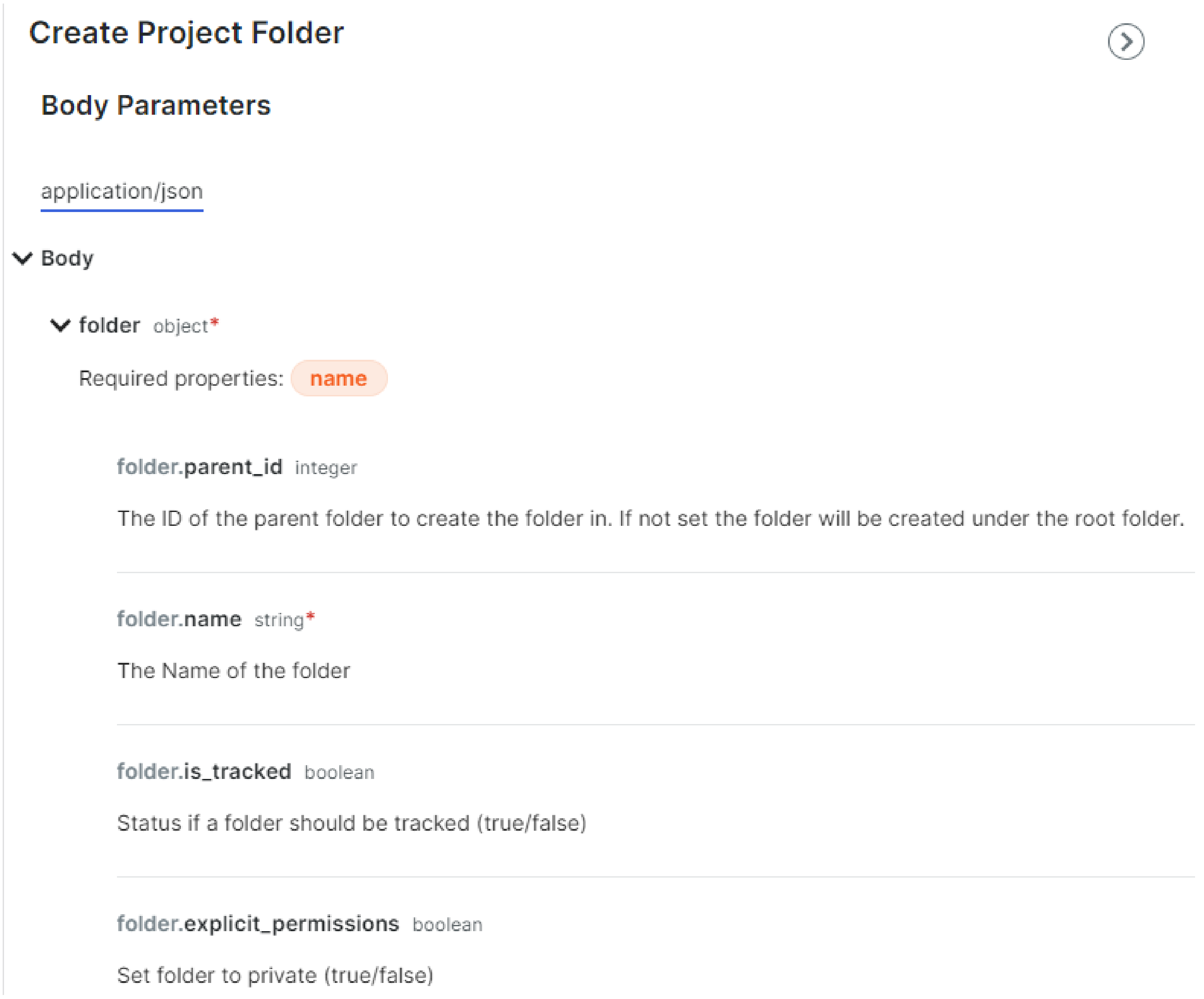
Slide 37: Request Body

How documentation specifies the request body

API documentation typically displays request body notes by providing detailed information on how to structure and format the data that you need to send in the request body when making an API call. These notes may include:

- **Data Format**: Documentation explains the expected data format, which can be JSON, XML, form data, or another format.
- **Parameters**: It lists the parameters or fields that should be included in the request body. Each parameter is described, including its name, type, and whether it's required.
- **Example**: Documentation often provides an example request body, showing you exactly how the data should be structured.
- **Data Types**: It clarifies the data types allowed for each parameter (e.g., string, number, boolean) and may specify any constraints or validation rules.
- **Validation Rules**: Documentation may detail any specific validation rules or patterns that the data must adhere to, such as minimum/maximum lengths or allowed characters.
- **Default Values**: If some parameters have default values, those values are usually documented.
- **Notes and Descriptions**: You may find explanations or descriptions of each parameter, helping you understand their purpose and usage.

Below is a screenshot taken from the Procore API Documentation on how to create a folder in Procore using a POST request:


Create Project Folder

Body Parameters

application/json

Body

folder object*

Required properties: name

folder.parent_id integer

The ID of the parent folder to create the folder in. If not set the folder will be created under the root folder.

folder.name string*

The Name of the folder

folder.is_tracked boolean

Status if a folder should be tracked (true/false)

folder.explicit_permissions boolean

Set folder to private (true/false)

The actual JSON data that you would send in the body would look something like this:

```
{
  "folder": {
    "parent_id": 12,
    "name": "test_folder",
    "is_tracked": true,
    "explicit_permissions": true,
    "custom_field_%{custom_field_definition_id}": "string"
  }
}
```

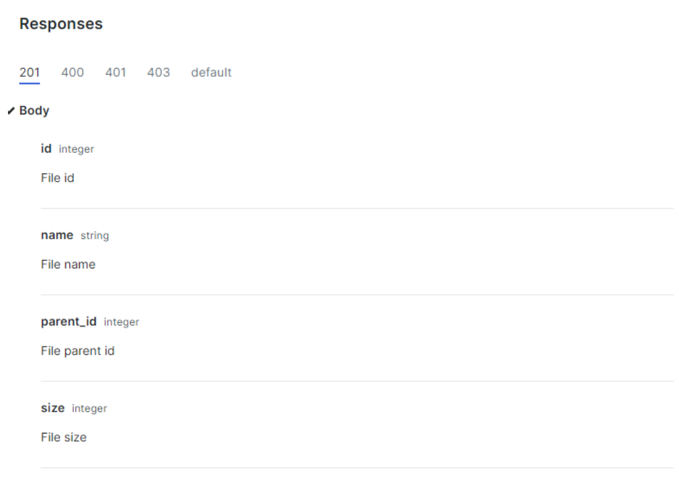
Slide 38: Responses

Example responses bodies and status codes

Documentation typically contains:

- **Status Codes**: Documentation lists common HTTP status codes that the API request can return.
- **Response Descriptions**: For each status code, documentation provides a description of what it means in the context of the API.

- **Example Responses**: Documentation may include sample API responses for each status code. These examples show the structure and content of the response body that you can expect to receive in different scenarios.
- **Response Formats**: Information about the response format, such as JSON or XML, is usually provided alongside the examples to ensure developers know how to parse and work with the response data.

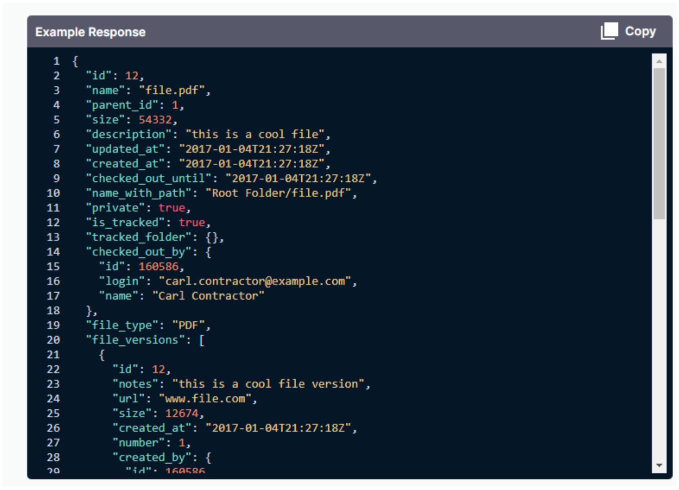


id integer
File id

name string
File name

parent_id integer
File parent id

size integer
File size



```
{
  "id": 12,
  "name": "file.pdf",
  "parent_id": 1,
  "size": 54332,
  "description": "this is a cool file",
  "updated_at": "2017-01-04T21:27:18Z",
  "created_at": "2017-01-04T21:27:18Z",
  "checked_out_until": "2017-01-04T21:27:18Z",
  "name_with_path": "Root Folder/file.pdf",
  "private": true,
  "is_tracked": true,
  "tracked_folder": {},
  "checked_out_by": {
    "id": 160586,
    "login": "carl.contractor@example.com",
    "name": "Carl Contractor"
  },
  "file_type": "PDF",
  "file_versions": [
    {
      "id": 12,
      "notes": "this is a cool file version",
      "url": "www.file.com",
      "size": 12674,
      "created_at": "2017-01-04T21:27:18Z",
      "number": 1,
      "created_by": {
        "id": 160586
```

Some documenation, like for Procore's API, include the response by status code which can be helpful to debug issues.

Slide 39: Authentication

How to manage authentication for your requests

In API documentation, authentication is a crucial topic covered to guide developers on how to securely access the API. Here's how authentication is typically addressed in API documentation

- **Authentication Methods**: API documentation explains the available authentication methods, such as API keys, OAuth tokens, JWT (JSON Web Tokens), or basic authentication.
- **Authentication Endpoints**: For APIs using OAuth or token-based authentication, documentation provides information on the authentication endpoints, where developers can obtain the necessary tokens or keys.
- **Authentication Parameters**: Documentation specifies the required authentication parameters that must be included in API requests, such as API keys, tokens, or credentials. It details where to place these parameters, whether in headers, query parameters, or request bodies.
- **Example Authentication**: To demonstrate the authentication process, API documentation often includes step-by-step examples of how to obtain authentication credentials and use them in API requests.
- **Rate Limiting**: Some API documentation may mention rate limiting, which restricts the number of requests a client can make within a certain time frame.
- **Token Refresh**: If applicable, documentation explains how to refresh authentication tokens and manage token expiration.
- **Error Handling**: Documentation provides information on error responses related to authentication, helping developers troubleshoot authentication issues.

You can view the authentication documentation for a few APIs:

- Procore

- GitHub
- NASA

Break (10 minutes)

👐Hands-On: Making API Requests

▶ Click to Expand

Slide 41: Hands-On Agenda

During the Hands-On Session we will be:

1. Learning how to extract data from responses systematically
2. Use the Procore API

🏆 Challenge: Making a GET Request

Use the PokeAPI in Postman to make a request to get data on your favorite Pokemon!

1. Navigate to the PokeAPI documentation and find the endpoint that gives you the overall information on a Pokemon
2. Craft the GET request in Postman
3. Be the first to do so and win a prize!

Slide 43: Using Reponse Data in Postman

Use the links below to find more information:

- For RO: Playbook

Slide 44: Using Reponse Data in Postman

Use the links below to find more information:

- For RO: Playbook

Slide 45: Using Reponse Data in Postman

Use the links below to find more information:

- For RO: Playbook

Slide 46: Using Reponse Data in Postman

Use the links below to find more information:

- For RO: Playbook
