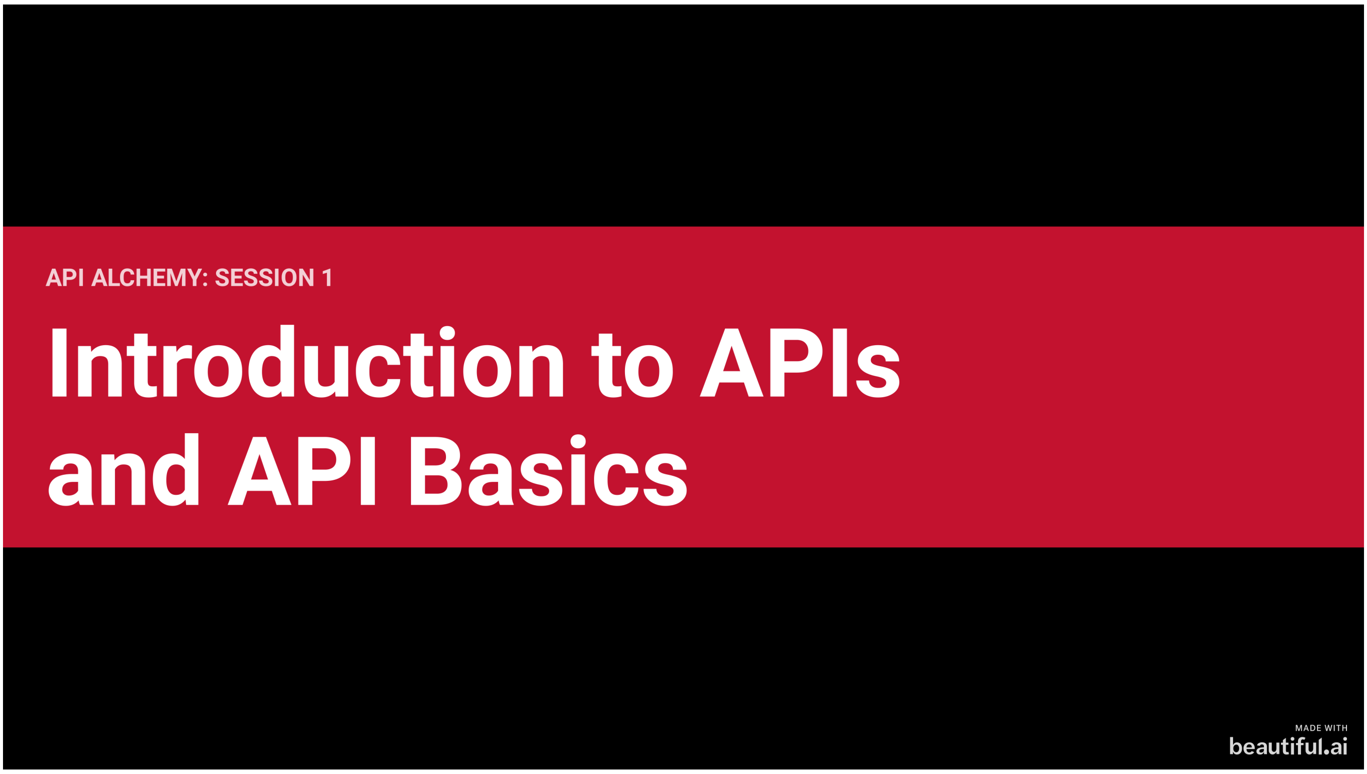
Session 1: Introduction to APIs and API Basics (2 hours)

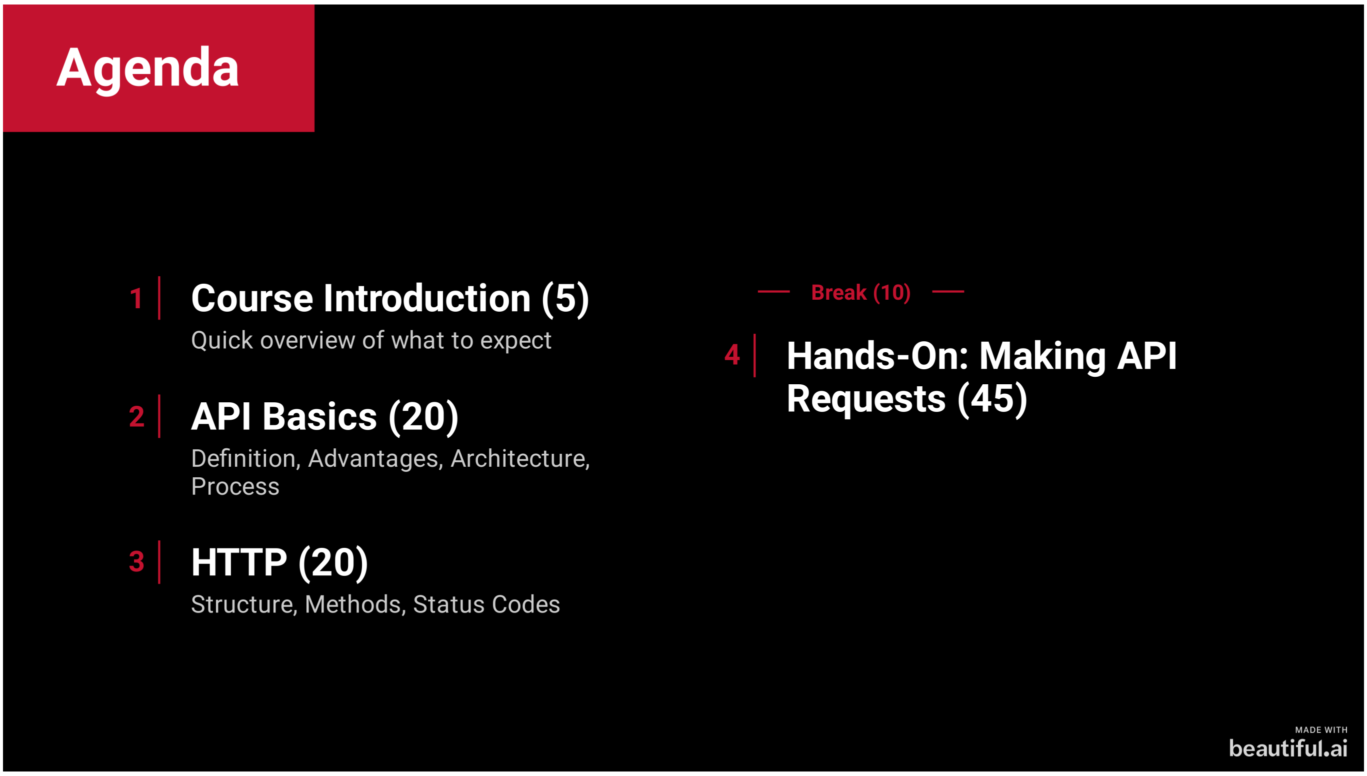
Gain an understanding of APIs and their role in modern software development, along with exploring the fundamental concepts of REST and SOAP methods.

Link to Presentation

Slide 1: Title



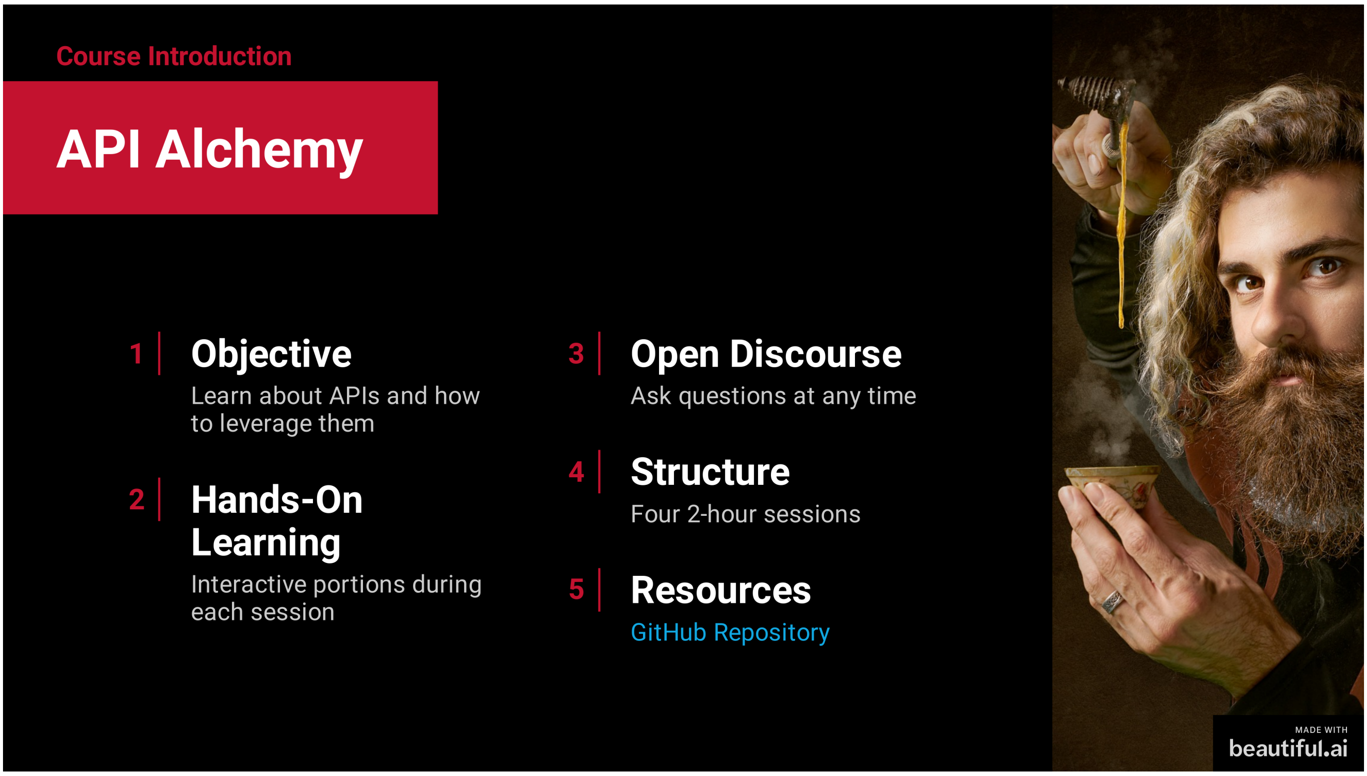
Slide 2: Agenda



- Provide an overview of the topics that will be discussed during this session
- Be sure to point out that numbers refer to the approximate time in minutes

Slide 3: Course Introduction

General overview of what to expect from the class



- **Objective**: Learn about what APIs are, what they can do for you, and how to leverage them
- **Hands-On Learning**: Stress that these sessions will always have a significant portion dedicated to hands-on, guided learning so that the topics can stick a bit more.

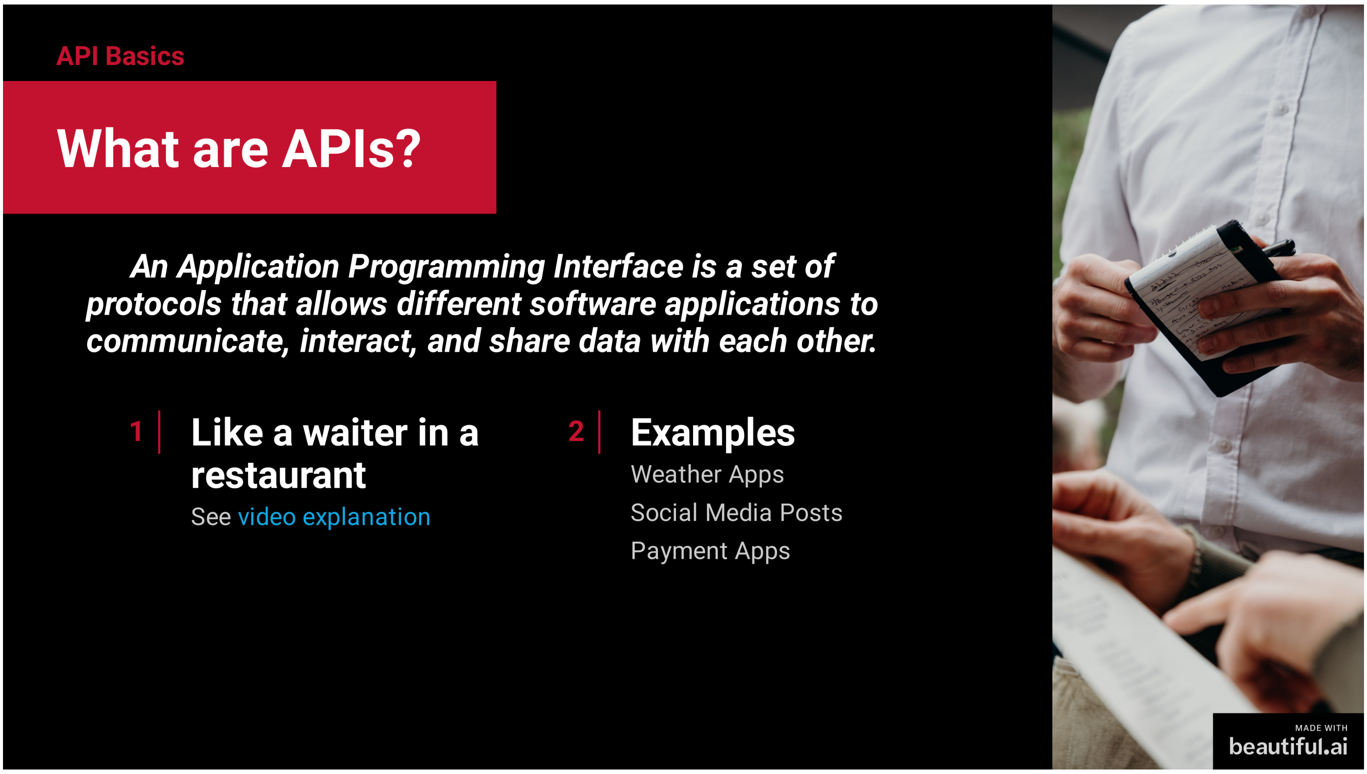
- **Open Discourse**: Questions are highly encouraged at any point, but be respectful to others if we are going over. Hagen is always available to discuss more on a topic outside of the session. If an answer is not known, Hagen will find it and share with the class at a later time.
- **Structure**: Training is broken into four 2-hour sessions with a 10-minute break provided around the midpoint of the session.
- **Resources**: Session agenda, notes, codes, and slides can be found on the GitHu repository page.

API Basics

▶ Click to Expand

Slide 5: Definition of APIs

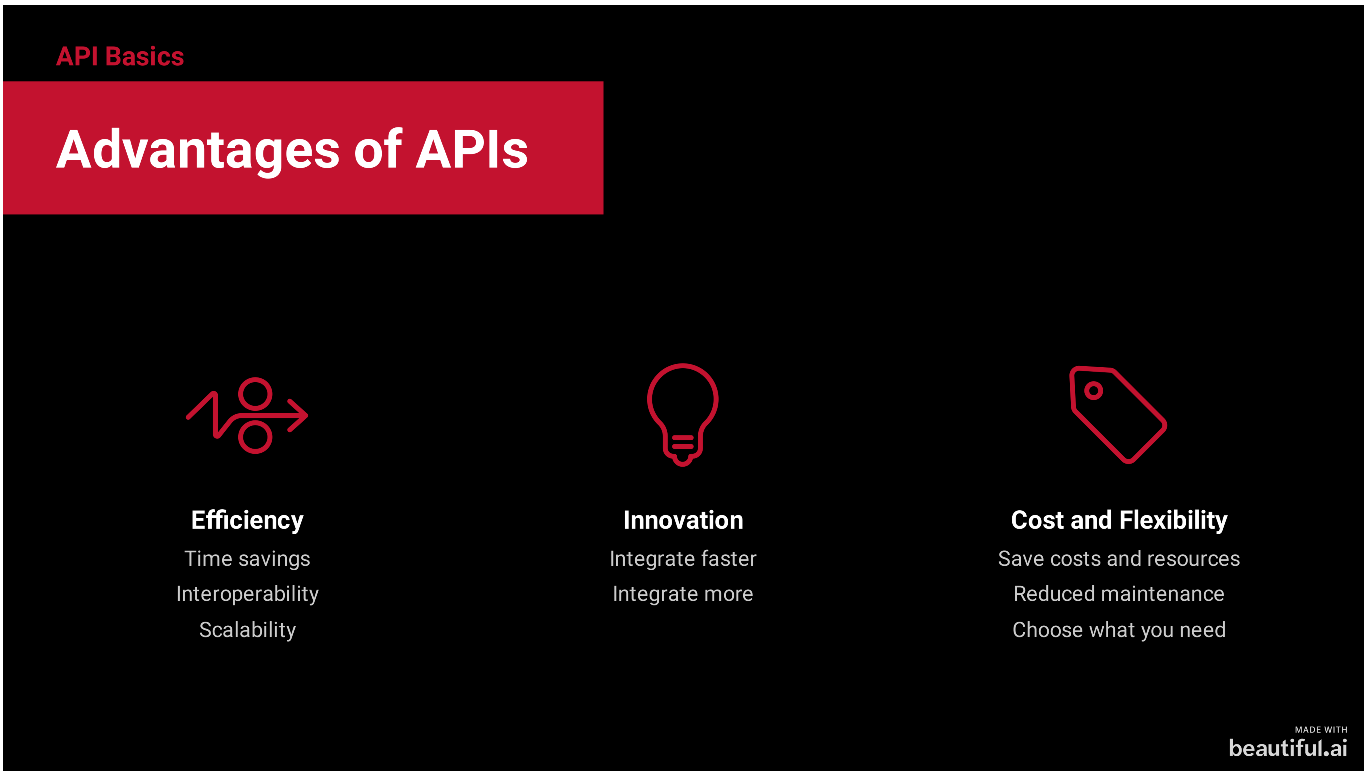
Overview of what an API is with examples



> 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
- 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

Slide 6: Advantages of APIs



Efficiency

- **Time savings**: No manual data entry and no manual code development. Simply use the API to pull the data you need.
- **Interoperability**: APIs allow you to interface with the main platform using a variety of applications. You can use almost any popular programming language and there are services that we will use later that make the job even easier.
- **Scalability**: You aren't storing the data on your device or in your application which allows your application to run faster and you don't have to deal with paying for storage.

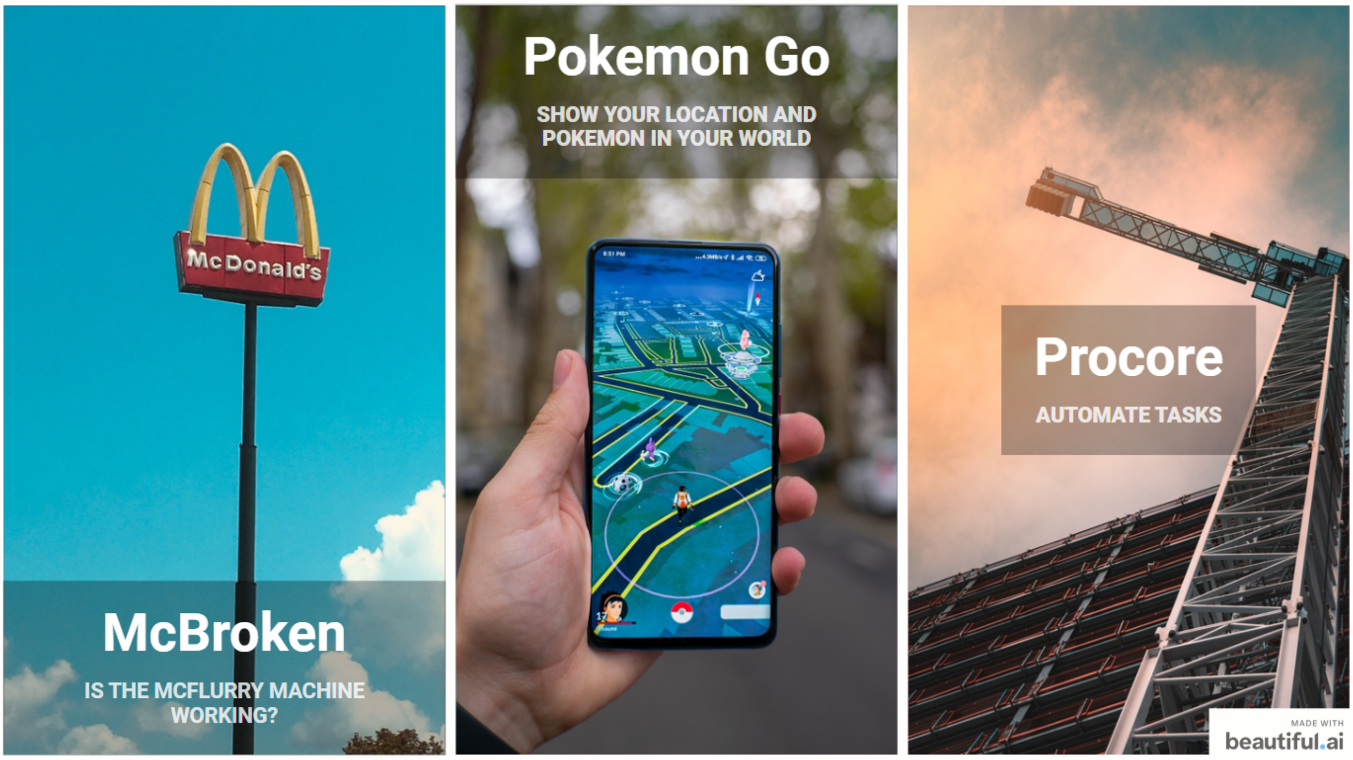
Innovation

- **Integrate faster**: You are relying on other people who are likely more experienced software developers to create pathways to the data you want. Once those pathways are created, you can quickly push and/or pull the data you need. You can also leverage their platform to get things like continuous and real-time updates.
- **Integrate more**: You can scale up the amount of information you pull into your application by using APIs from a wide variety of platforms to provide a richer experience.

Cost and Flexibility

- **Save costs and resources**: You can focus on the analysis side of things rather than the data gathering. Or you can discover ways to use the API to speed up your process that aren't available on the UI.
- **Reduced maintenance**: If you created your process to push or pull data, you would personally need to maintain that. If you use APIs, the platform will ensure that these pathways remain stable even if they change how their front- or back-end works.
- **Choose what you need**: API pathways are generally specific to what functions they can accomplish. Rather than doing bulk actions that could be time-consuming or provide excessive information, you can do exactly what you like.

Slide 7: 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

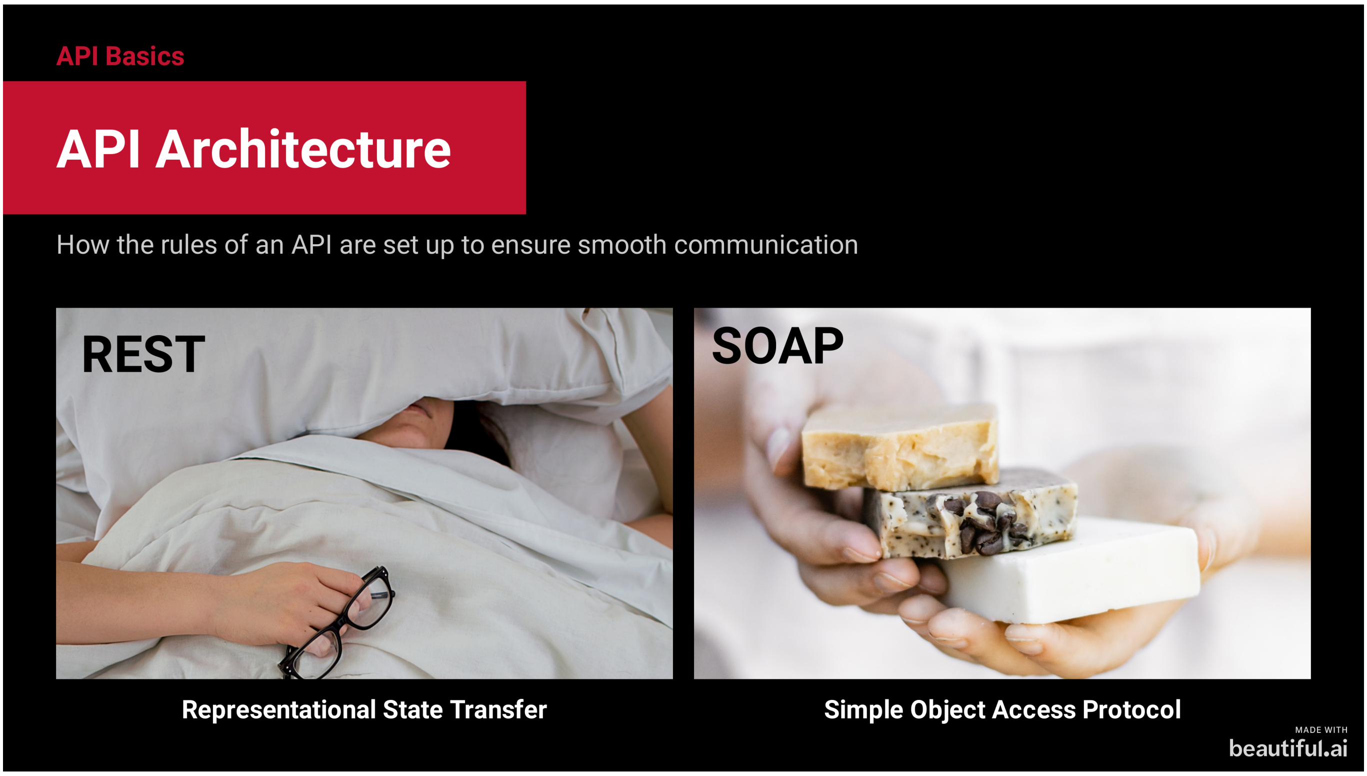
Procore Permissioning

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
- 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).

Slide 8: API Architecture

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



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

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

Slide 9: SOAP Overview

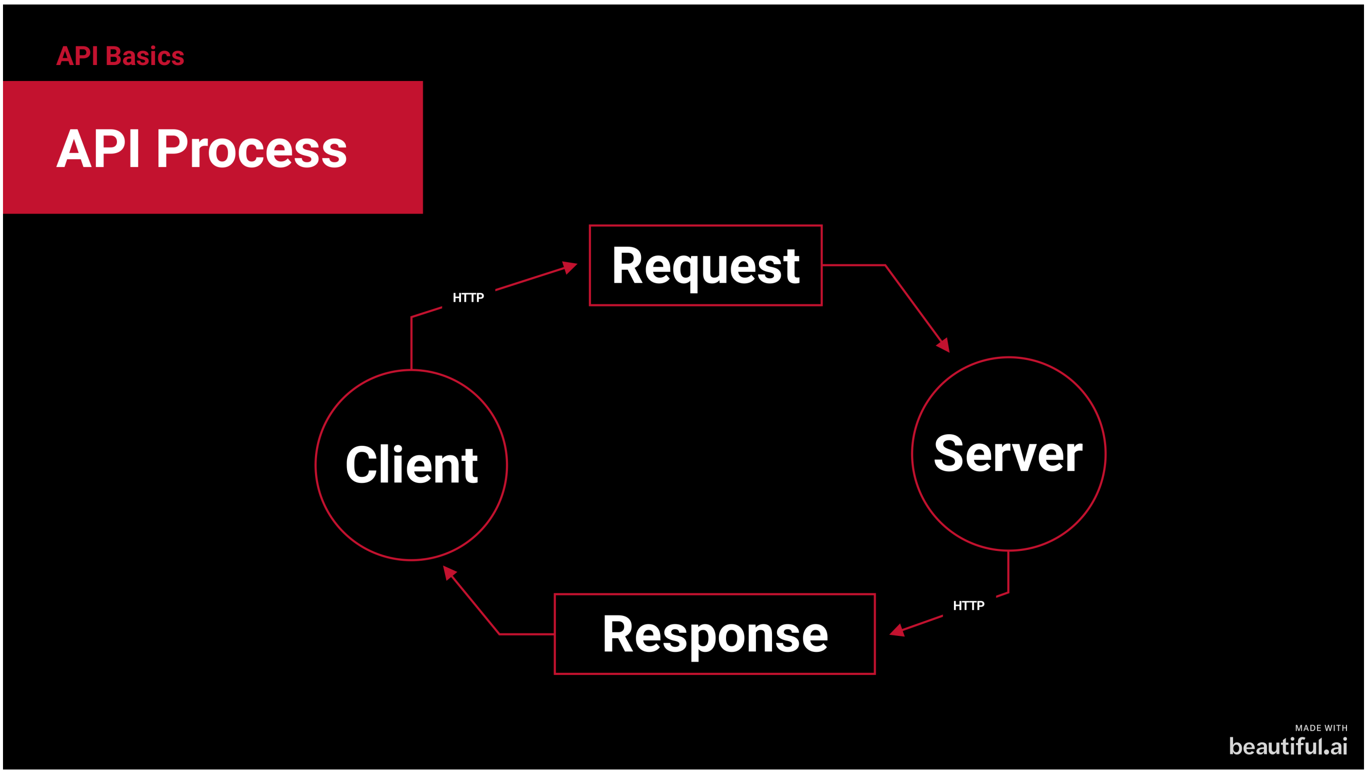
- **Principle**: Uses more formal rules, like sending a detailed package with instructions.
- **Use Cases**: Suited for complex applications, often used in big businesses and industries where strict communication is needed.
- **Advantages**: Structured and secure. Provides strict standards for messaging and security, suitable for enterprise scenarios.
- **Disadvantages**: Heavier and more complex compared to REST. May not be suitable for lightweight applications.

Slide 10: REST Overview

- **Principle**: Uses simple rules to communicate over the internet, like talking to a waiter to order food.
- **Use Cases**: Best for simpler applications like mobile apps and websites, where quick communication is important.
- **Advantages**: Easy to understand, lightweight, and flexible. Works well for microservices and modern web applications.
- **Disadvantages**: Less structured than SOAP, not ideal for complex enterprise-level applications.

Slide 11: Process Overview

General process when invoking an API



Client-Server Architecture: The API process relies on a client-server model where the client makes requests and the server processes and responds.

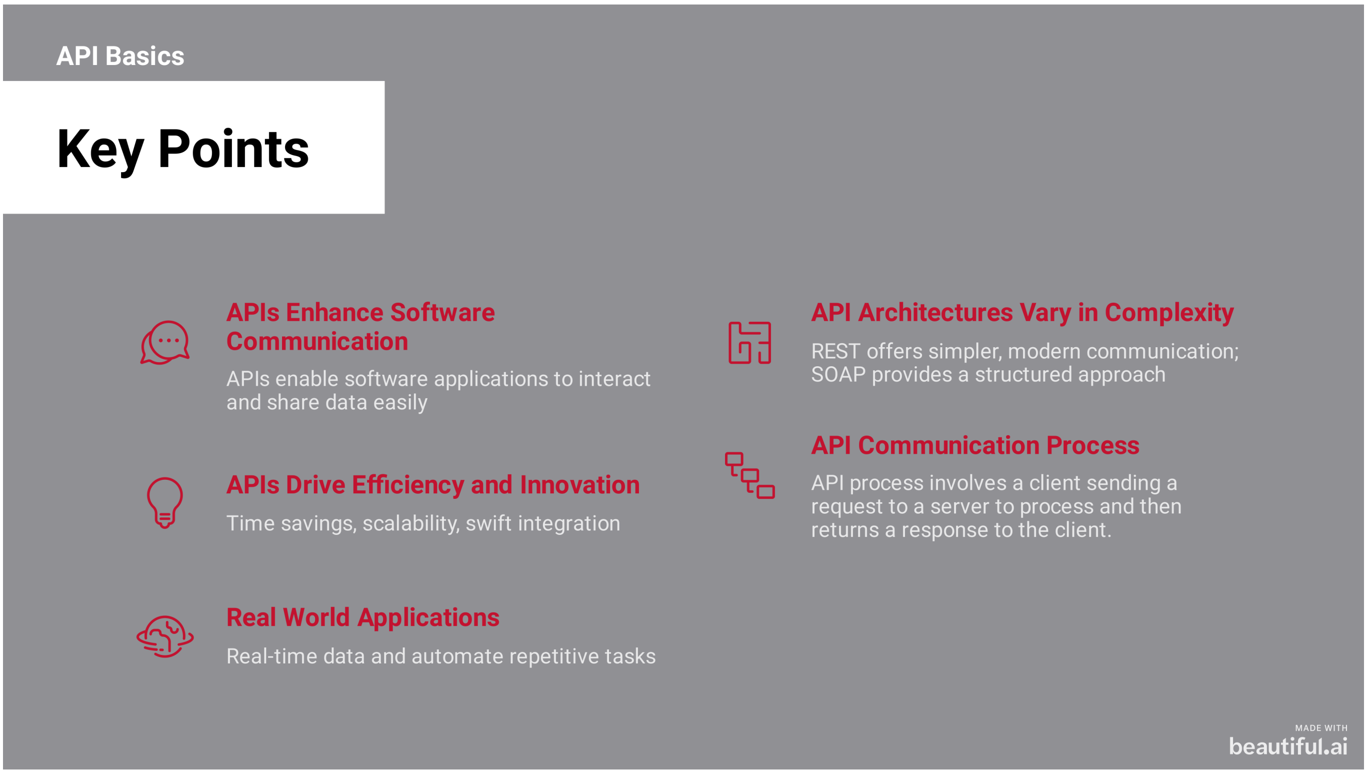
1. **Request**:
   - Initiating Point: The client sends a request using an HTTP method
   - Contains Data: In some cases, the request carries data (like user credentials, or the specifics of the data being requested)
2. **Server**:
   - Processing Center: The server processes the request, interacts with a databases or other necessary component, and creates an appropriate response
   - May Involve Logic: Depending on the request, the server might execute certain logic or computations before formulating a response
3. **Response**:
   - Feedback Mechanism: After processing, the server sends back a response which might contain data, confirmation of a successful operation, or an error message
   - Formats: Responses can come in various formats, commonly JSON or XML, which the client software can then interpret and display or use as needed
4. **HTTP**:

- Standard Protocol: HTTP is the foundation of data communication for the World Wide Web, used here to transfer requests and responses
- Status Codes: HTTP responses contain status codes that indicate the result of the request

5. **Cycle Continues**: Depending on the application and user actions, this process can happen repeatedly, ensuring real-time interaction and data retrieval

🔍 Key Points (Slide 12)

Summary from the API Basics section

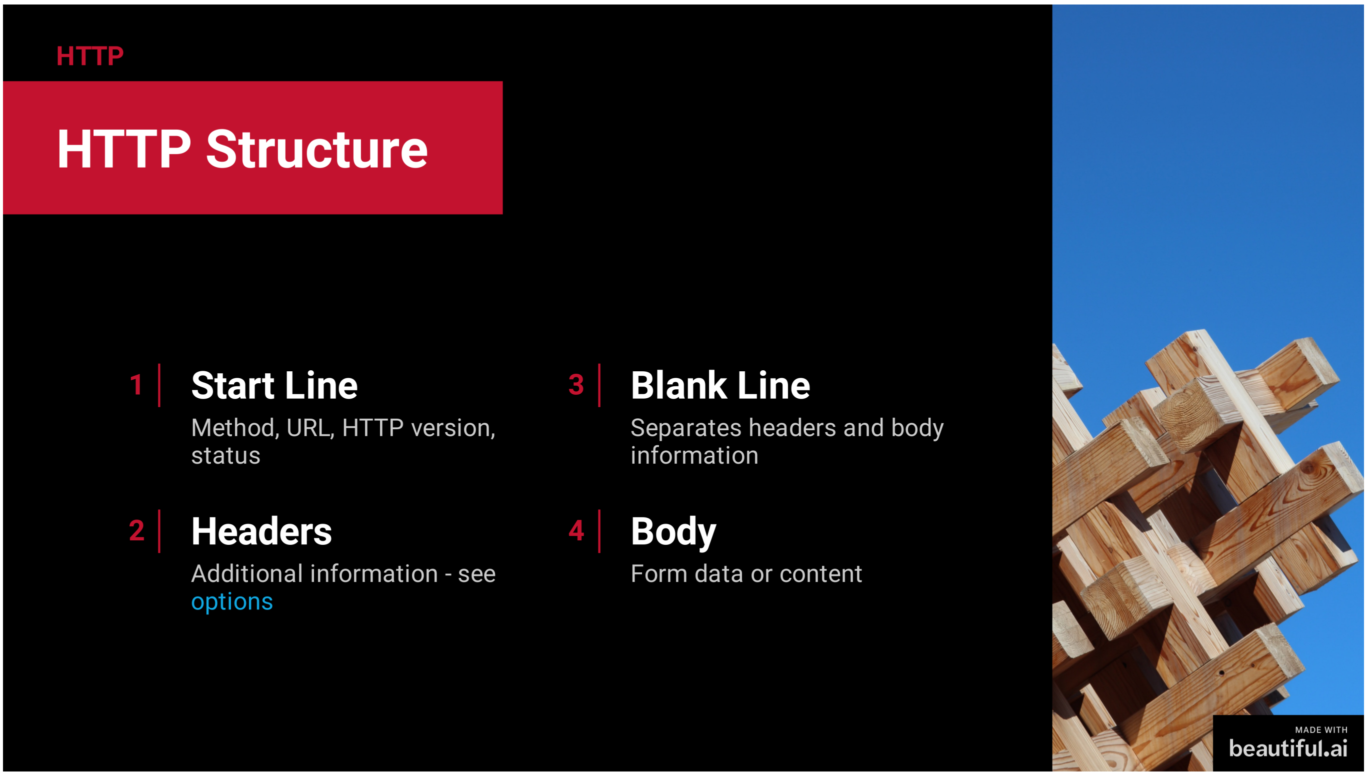


1. **APIs Enhance Software Communication**: Application Programming Interfaces (APIs) enable different software applications to interact and share data seamlessly, from weather updates to payment authorizations.
2. **APIs Drive Efficiency and Innovation**: They offer time savings, scalability, and swift integration capabilities, letting developers focus more on innovation and less on maintenance.
3. **Real-world API Applications**: Apps like McBroken and Pokémon Go utilize APIs for real-time data and augmented reality, while platforms like Procore automate repetitive tasks.
4. **API Architectures Vary in Complexity**: While REST offers a simpler, more modern communication method akin to ordering food, SOAP provides a structured approach resembling a detailed package with instructions.
5. **API Communication Process**: The typical API process involves a client sending a request to a server, which then processes the request and returns an appropriate response to the client. This interaction ensures timely and accurate data exchange between systems.

🌐 HTTP

▶ Click to Expand

Slide 14: 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

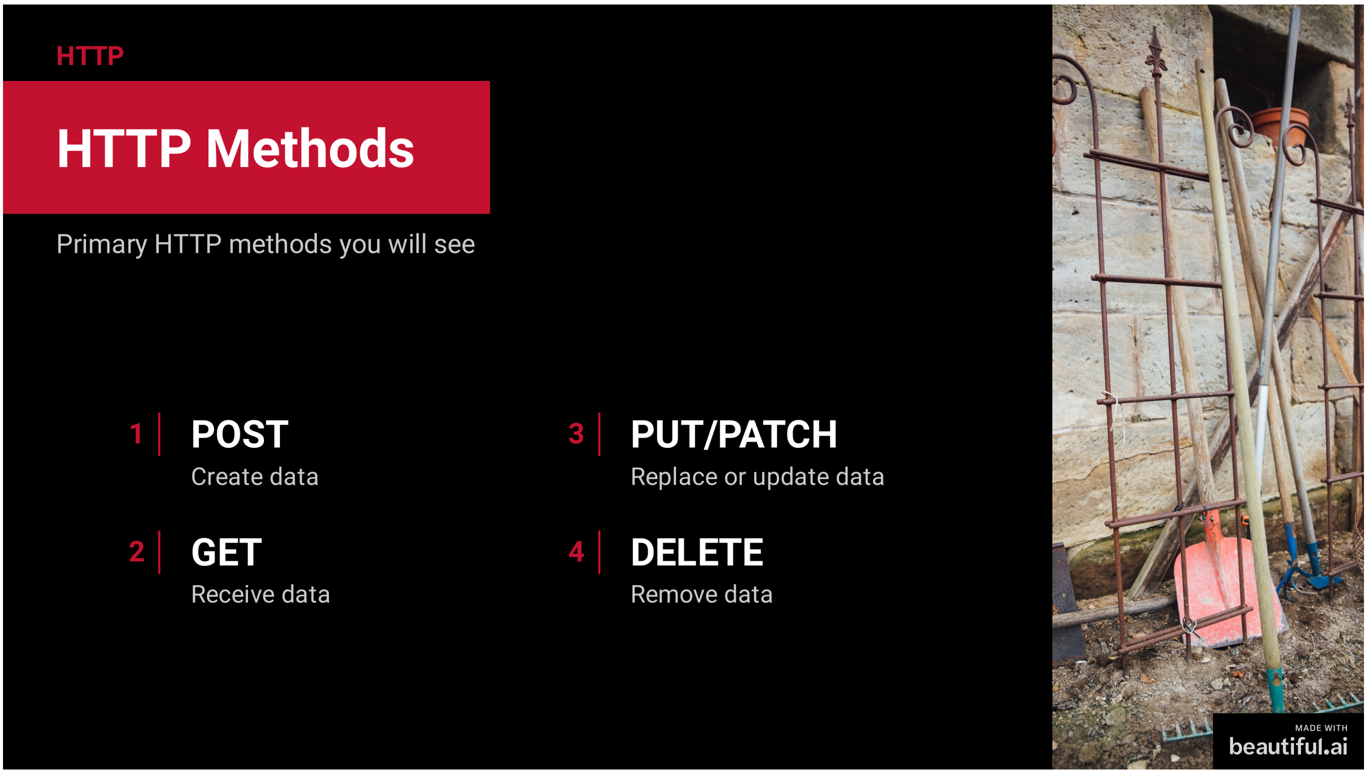
For responses, the body is often formatted in:

- JSON
- XML
- HTML

What are some HTTP Request Methods?

Slide 16: HTTP Methods

The data manipulation methods used by APIs



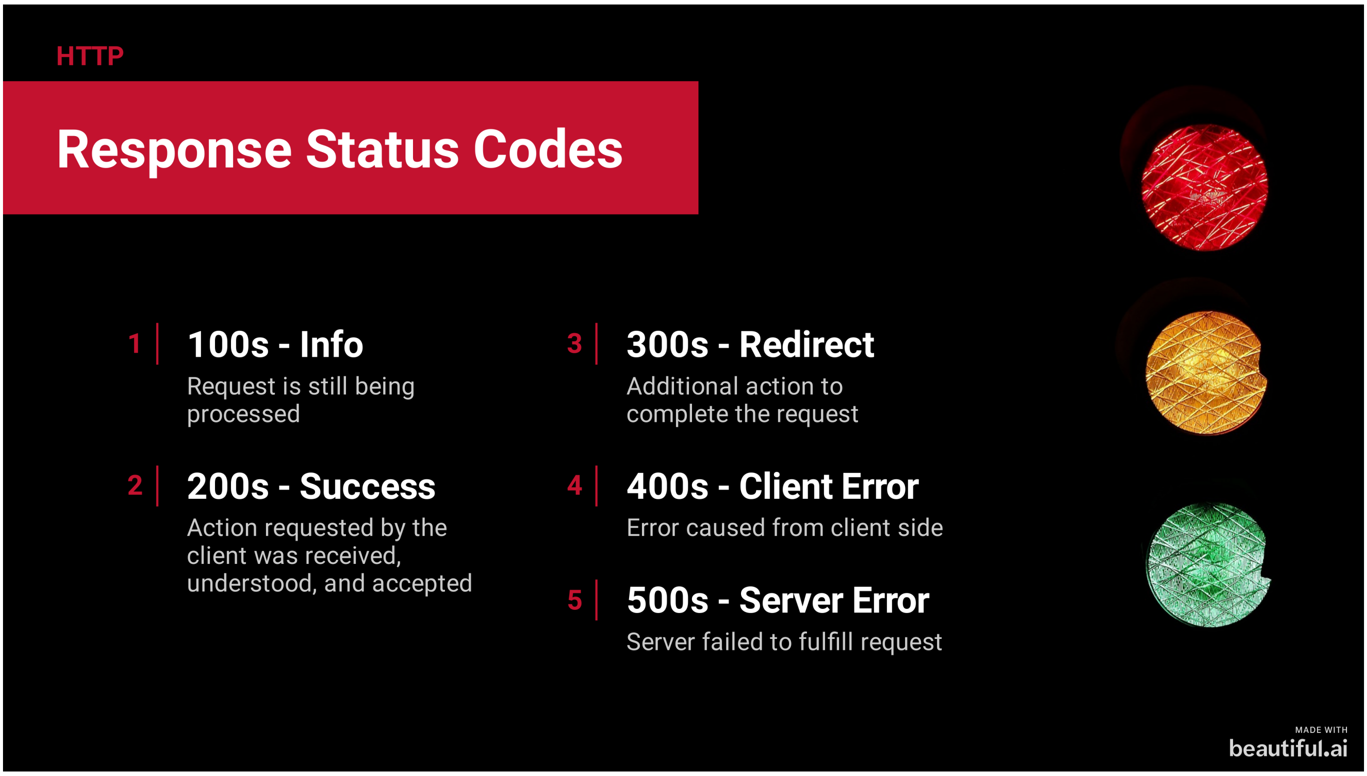
There are 9 HTTP methods in HTTP v1.1, but there are four/five ones that are commonly used:

- **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.

What are some HTTP Response Status Codes?

Slide 18: Response Status Codes

Basic breakdown of the status codes you might see when making API calls



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:

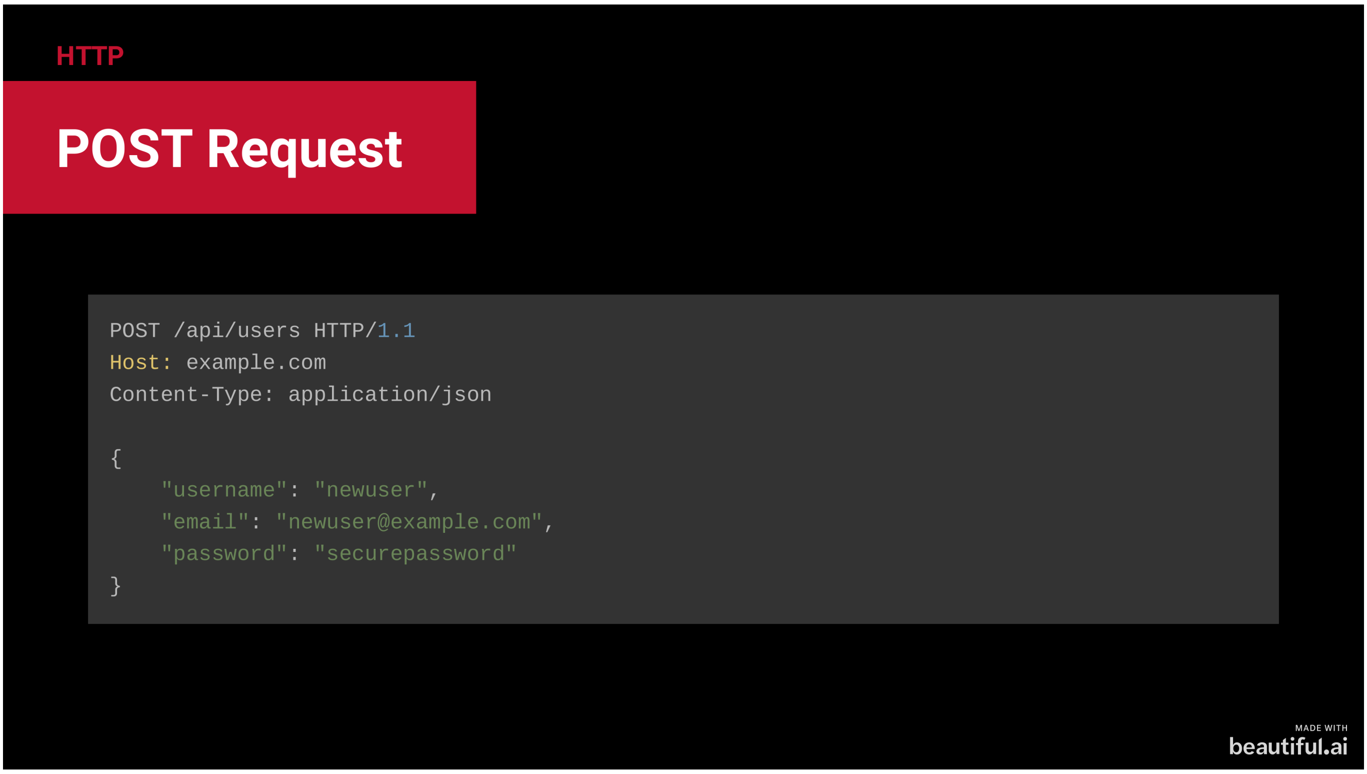
- **400 Bad Request**: The server cannot or will not process the request due to an apparent client error (bad request syntax, size too large, invalid request message framing, or deceptive request routing)

- **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.

Slide 19: POST Request




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

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

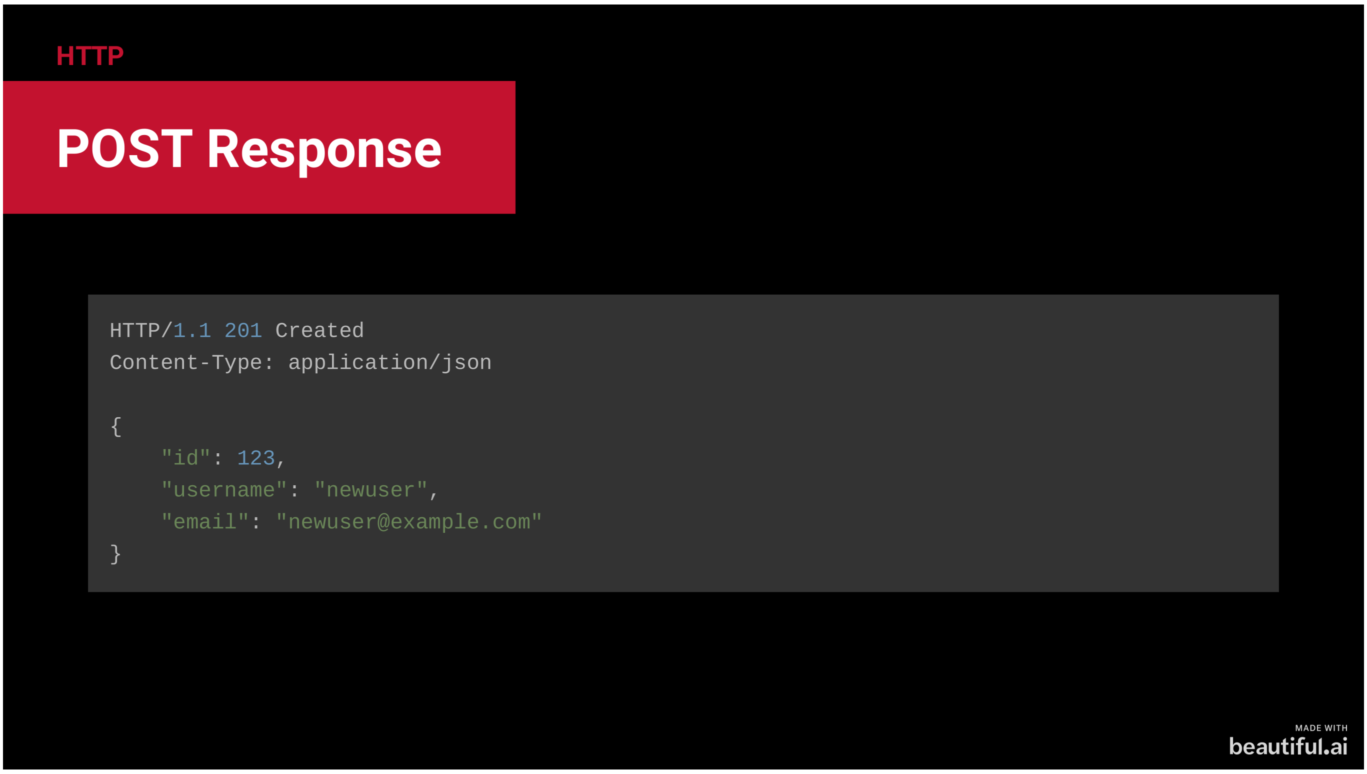
Identify the key components:

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 20: POST Response



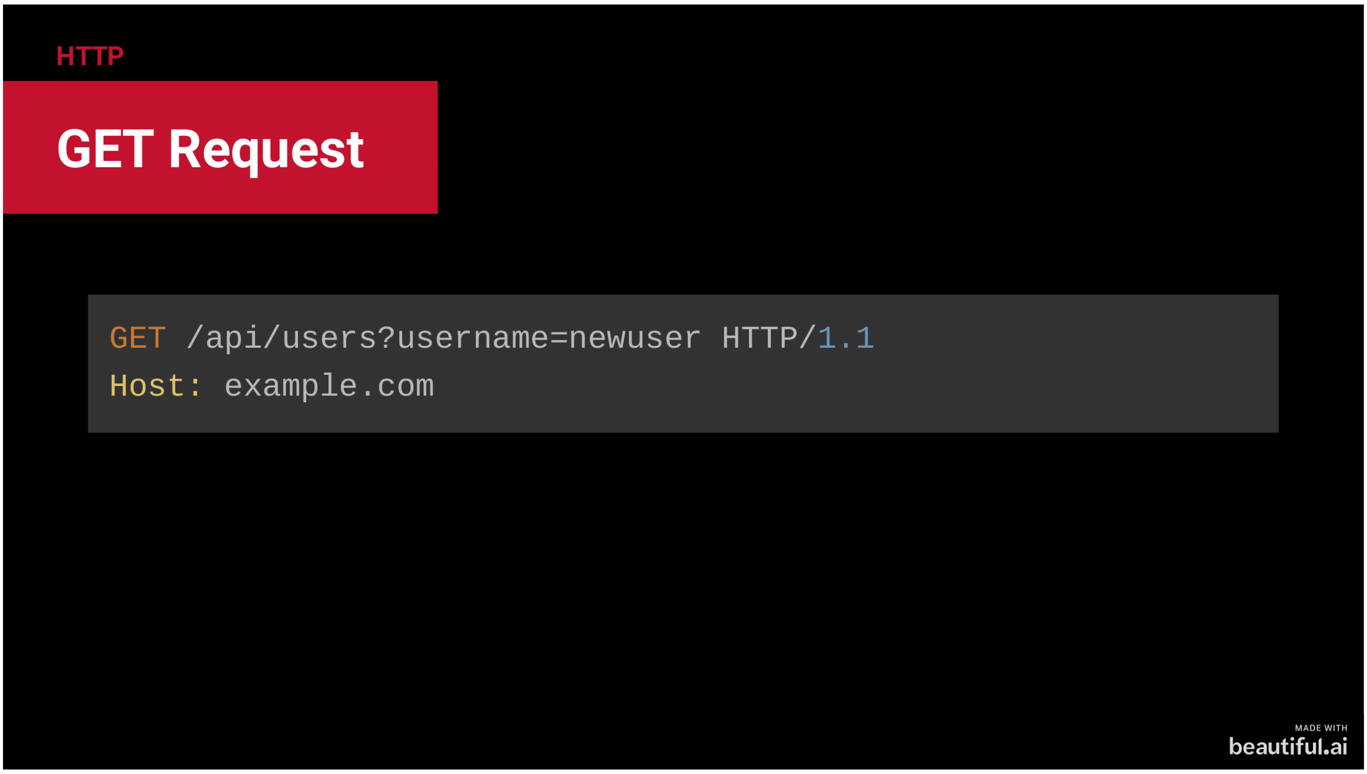

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "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 21: GET Request




```
GET /api/users?username=newuser HTTP/1.1
Host: example.com
```

Identify the key components:

1. Request Line
   - Method: `GET`
   - URL: `/api/users`

- Query Parameters: `?username=newuser`
- HTTP Version: `HTTP/1.1`

2. Headers
    - Header 1: `Host: example.com`

Important: We cannot include a body in a GET request so if we need to specify additional information, we do so through the use of query parameters included in the URL.

Slide 22: GET Response


```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 123,
    "username": "newuser",
    "email": "newuser@example.com",
    "bio": "User's biography"
}
```

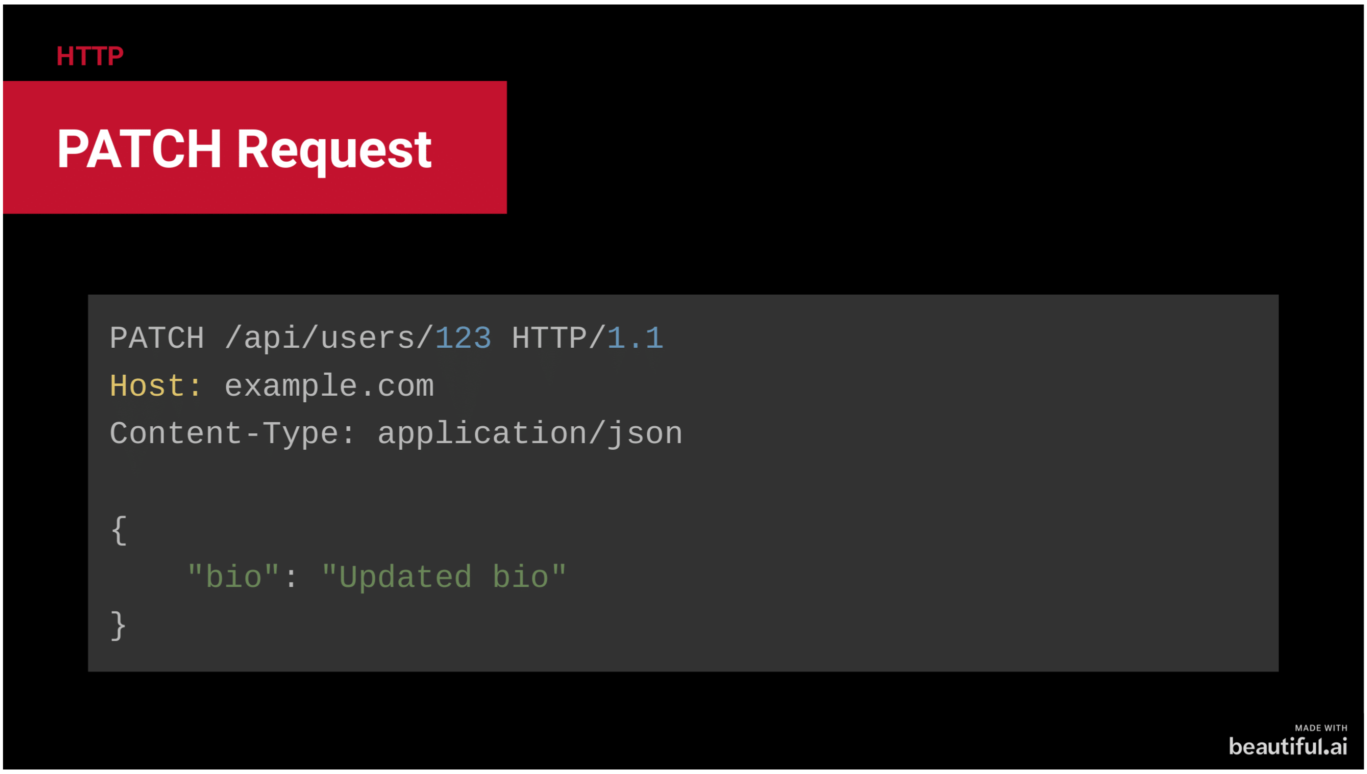
Identify the key components:

1. Status Line
    - HTTP Version: `HTTP/1.1`
    - Status Code: `200`
    - Status Text: `OK`
2. Headers
    - Header 1: `Content-Type: application/json`
3. Blank Line

4. Body
    - JSON Form:

    ```
    {
        "id": 123,
        "username": "newuser",
        "email": "newuser@example.com",
        "bio": "User's biography"
    }
    ```

Slide 23: PATCH Request




```
PATCH /api/users/123 HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "bio": "Updated bio"
}
```

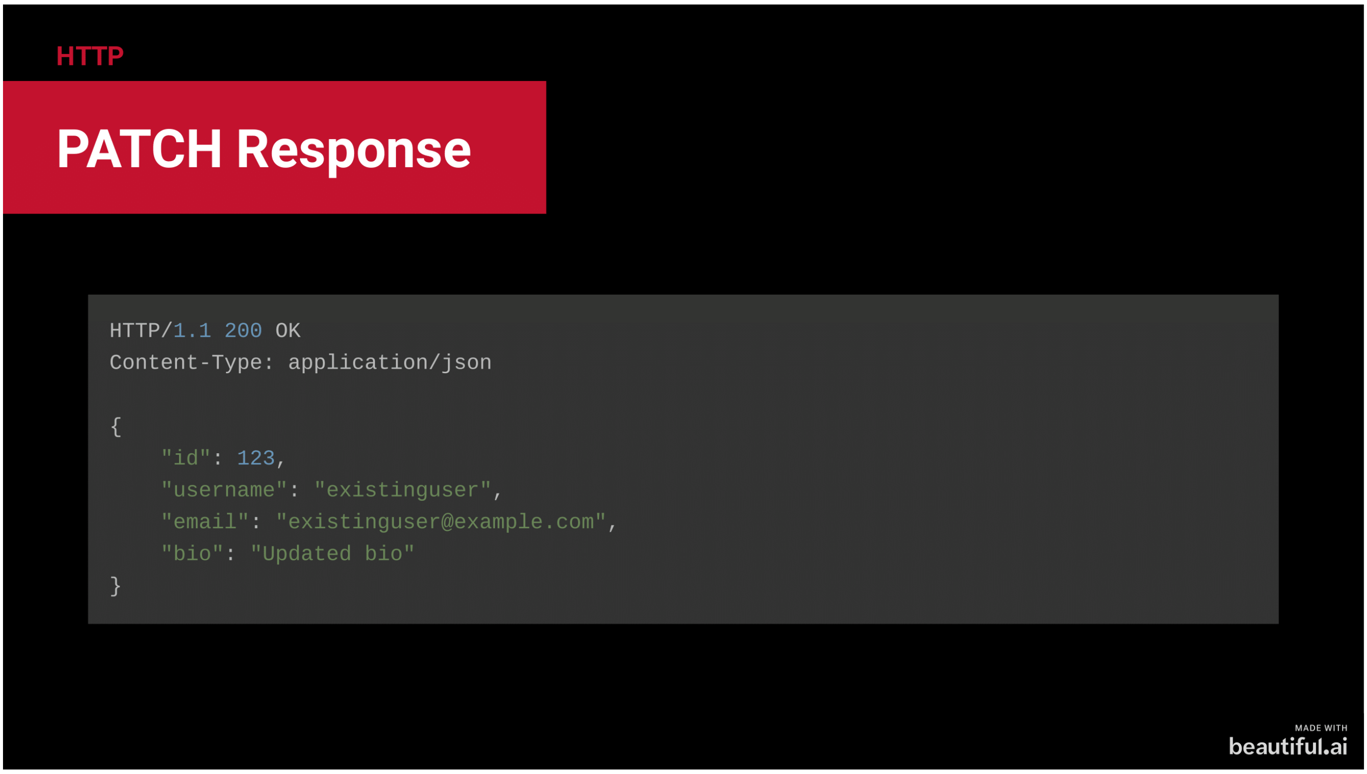
Identify the key components:

1. Request Line
    - Method: `PATCH`
    - URL: `/api/users/123`
    - 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:

    ```
    {
        "bio": "Updated bio"
    }
    ```

Slide 24: PATCH Response




```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 123,
    "username": "existinguser",
    "email": "existinguser@example.com",
    "bio": "Updated bio"
}
```

Identify the key components:

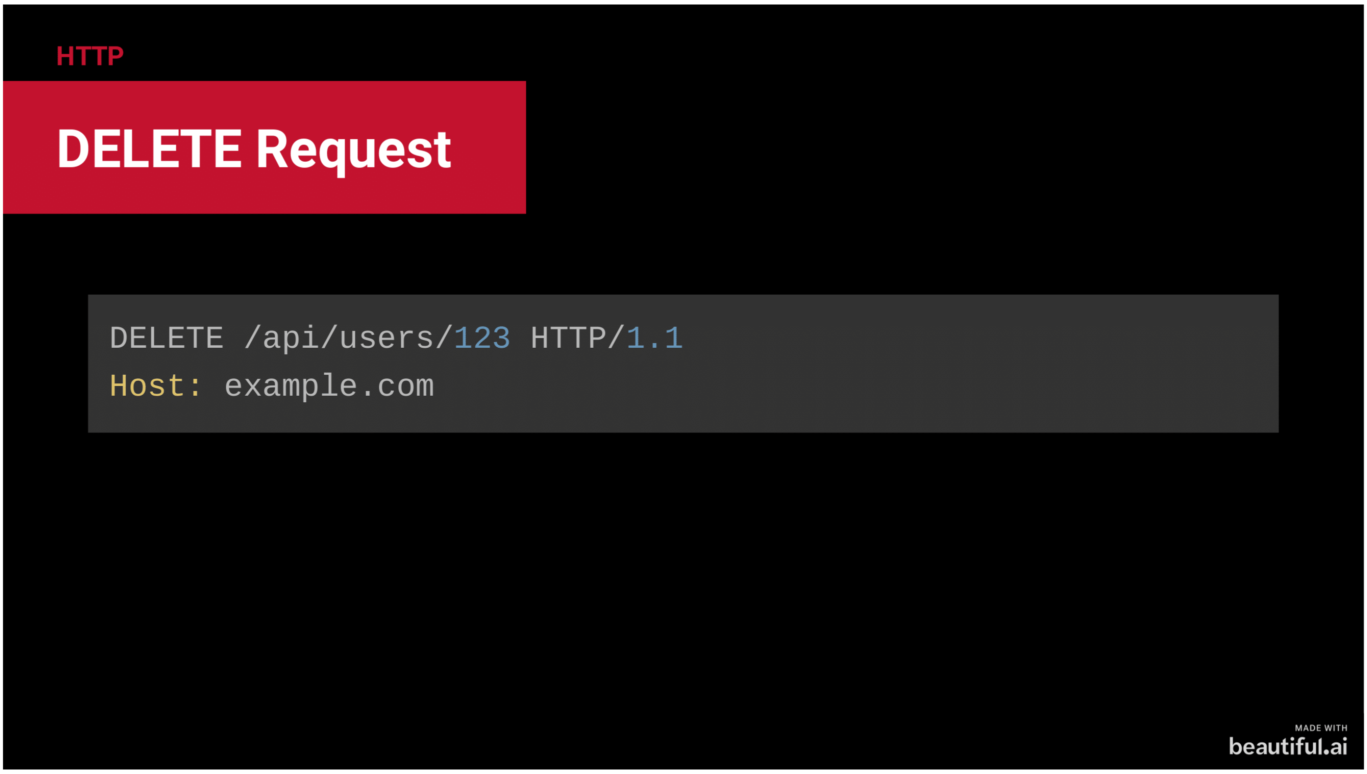
1. Status Line
    - HTTP Version: `HTTP/1.1`
    - Status Code: `200`

- Status Text: `OK`

2. Headers
   - Header 1: `Content-Type: application/json`
3. Blank Line
4. Body
   - JSON Form:

```
{
    "id": 123,
    "username": "existinguser",
    "email": "existinguser@example.com",
    "bio": "Updated bio"
}
```

Slide 25: DELETE Request




```
DELETE /api/users/123 HTTP/1.1
Host: example.com
```

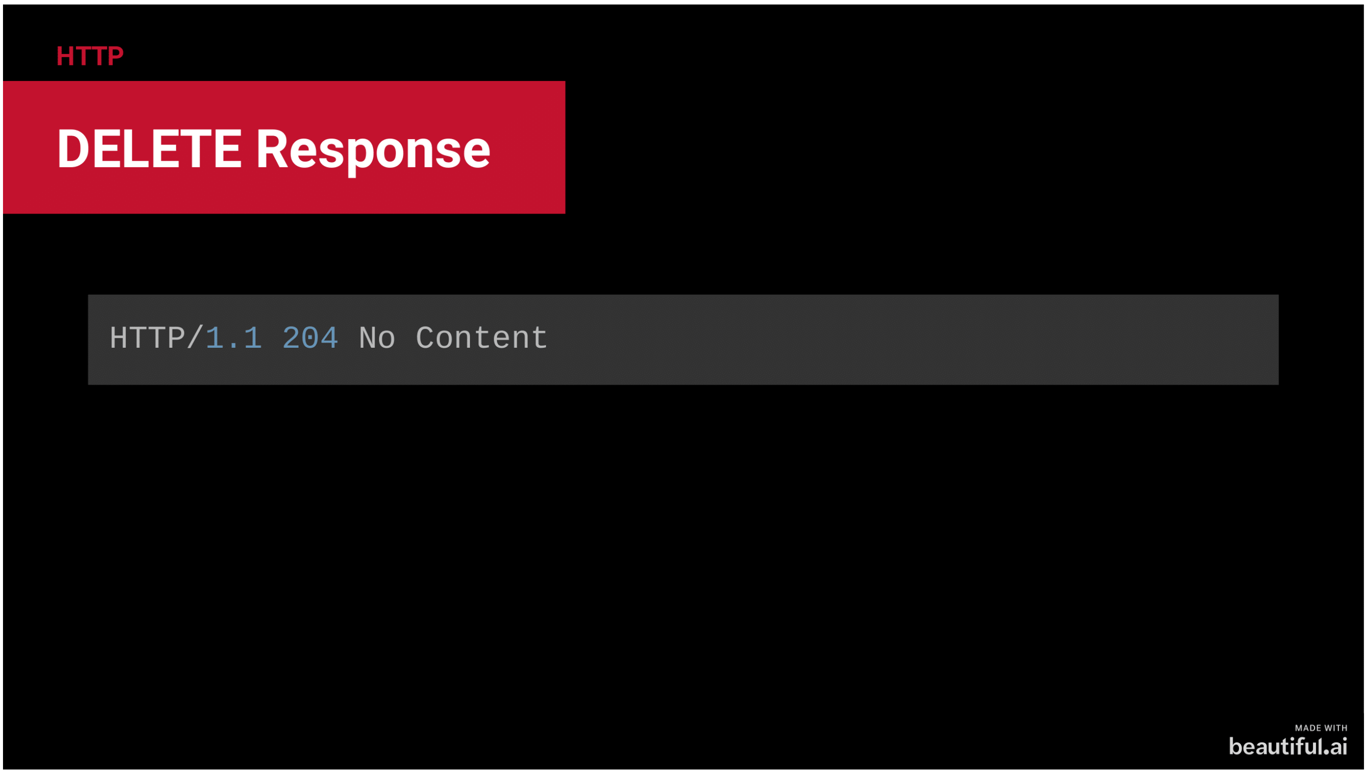
Identify the key components:

1. Request Line
   - Method: `DELETE`
   - URL: `/api/users/123`
   - HTTP Version: `HTTP/1.1`
2. Headers

- Header 1: `Host: example.com`

❗ **Important**: As with GET requests, we cannot include a body for DELETE requests.

Slide 26: DELETE Response



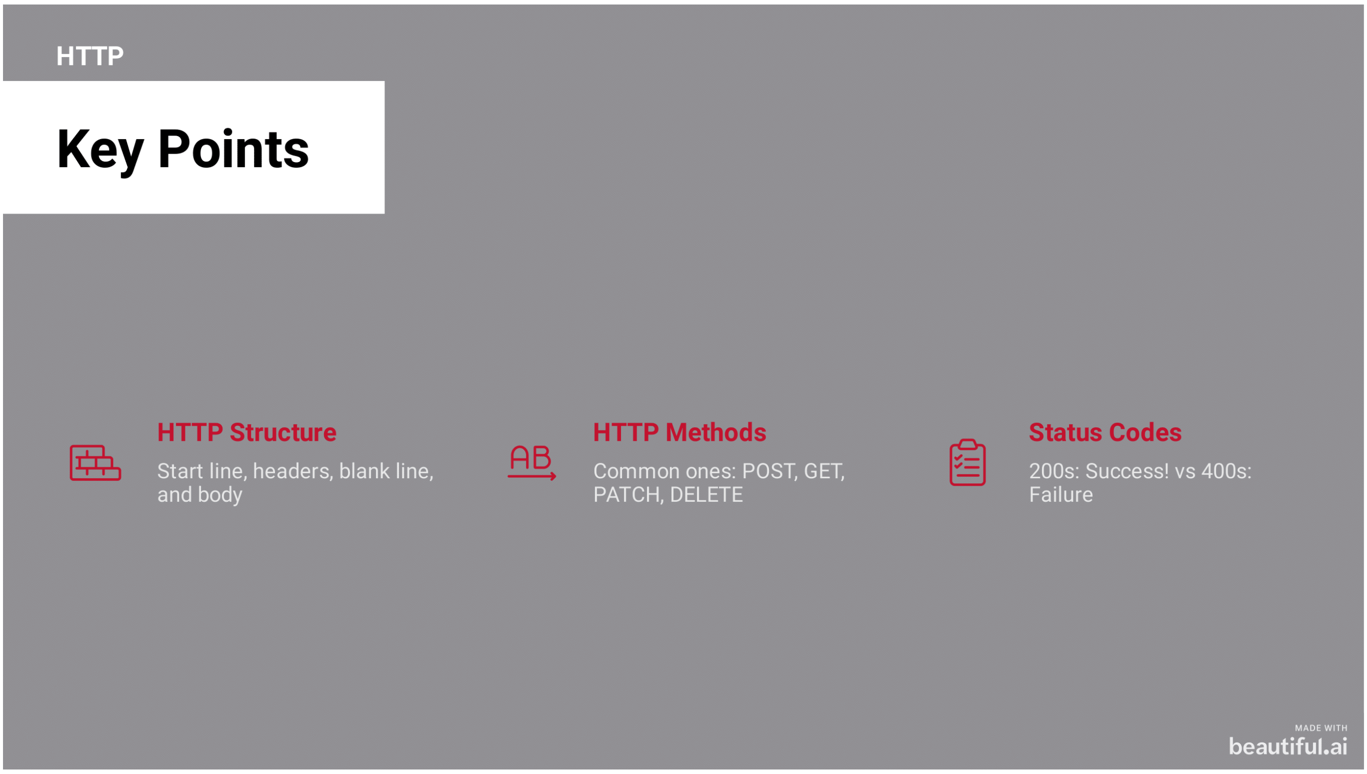

```
HTTP/1.1 204 No Content
```

Identify the key components:

1. Status Line
    - HTTP Version: `HTTP/1.1`
    - Status Code: `204`
    - Status Text: `No Content`

🔑 Key Points (Slide 27)

Summary from the HTTP section



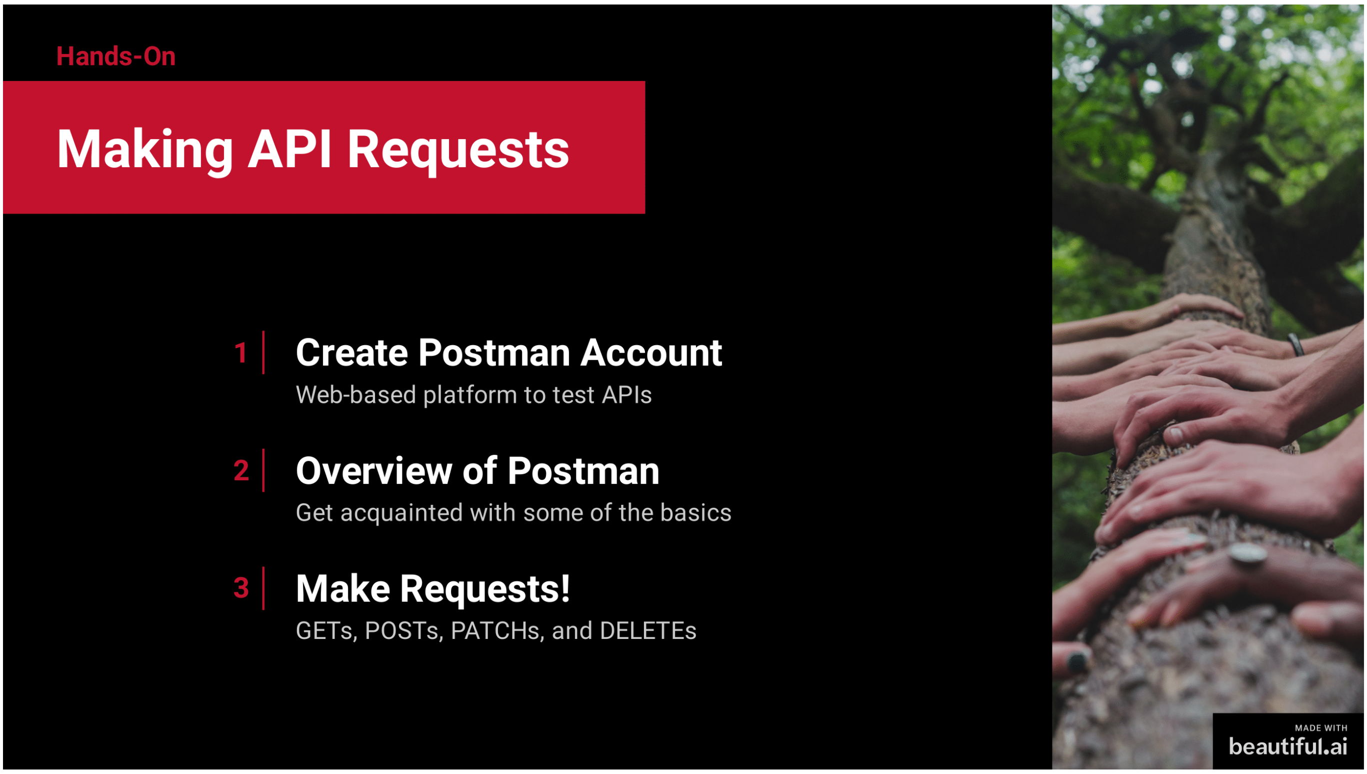
1. **HTTP Structure**:
   - Start Line: Contains the HTTP method, URL, parameters, and protocol version for requests, and status code, description, and protocol version for responses.
   - Headers: Provide additional standardized information, such as Authorization, Content-Type, and Host.
   - Blank Line: Differentiates headers from the body.
   - Body: Carries additional data, commonly formatted in JSON, XML, or HTML.
2. **HTTP Methods**: There are many, but the primary ones you will likely deal with are:
   - POST: Sends data to the server.
   - GET: Retrieves information using a URI.
   - PUT: Replaces all representations of the target resource.
   - PATCH: Partially modifies a resource.
   - DELETE: Removes the specified resource.
3. **Response Status Codes**: Look out pimrarily for the 200s 🥰 or the 400s 🤬
   - 100s: Informational responses.
   - 200s: Indicate success, e.g., 200 OK, 201 Created.
   - 300s: Additional steps, often related to URL redirection.
   - 400s: Client-side errors, e.g., 400 Bad Request, 401 Unauthorized.
   - 500s: Server-side errors.
4. **Notable Pointers**:
   - Bodies are not included in GET and DELETE requests.
   - Additional information for GET requests is passed using query parameters in the URL.

Break (10 minutes)

🤲 Hands-On: Making API Requests

▶ Click to Expand

Slide 29: Hands-On Agenda



During the Hands-On session, we will be:

1. Creating a Postman Account
2. Getting an Overview of the Postman Platform
3. Making API Requests!

Slide 30: Creating Postman Account

Use the links below to find more information:

- For RO: Playbook
- For Others: GitHub

Slide 31: Workspaces and Concepts

Use the links below to find more information:

- For RO: Playbook
- For Others: GitHub

Slide 32: Collections

Use the links below to find more information:

- For RO: Playbook
- For Others: GitHub

Slide 33: GET Requests

Use the links below to find more information:

- For RO: Playbook
- For Others: GitHub

Slide 34: POST request

Use the links below to find more information:

- For RO: Playbook
- For Others: GitHub

Slide 35: PATCH request

Use the links below to find more information:

- For RO: Playbook
- For Others: GitHub

Slide 36: DELETE request

Use the links below to find more information:

- For RO: Playbook
- For Others: GitHub
