Lecture Week 04: Web Services

Adopted from Notes of Dr. Hamed Hamzeh

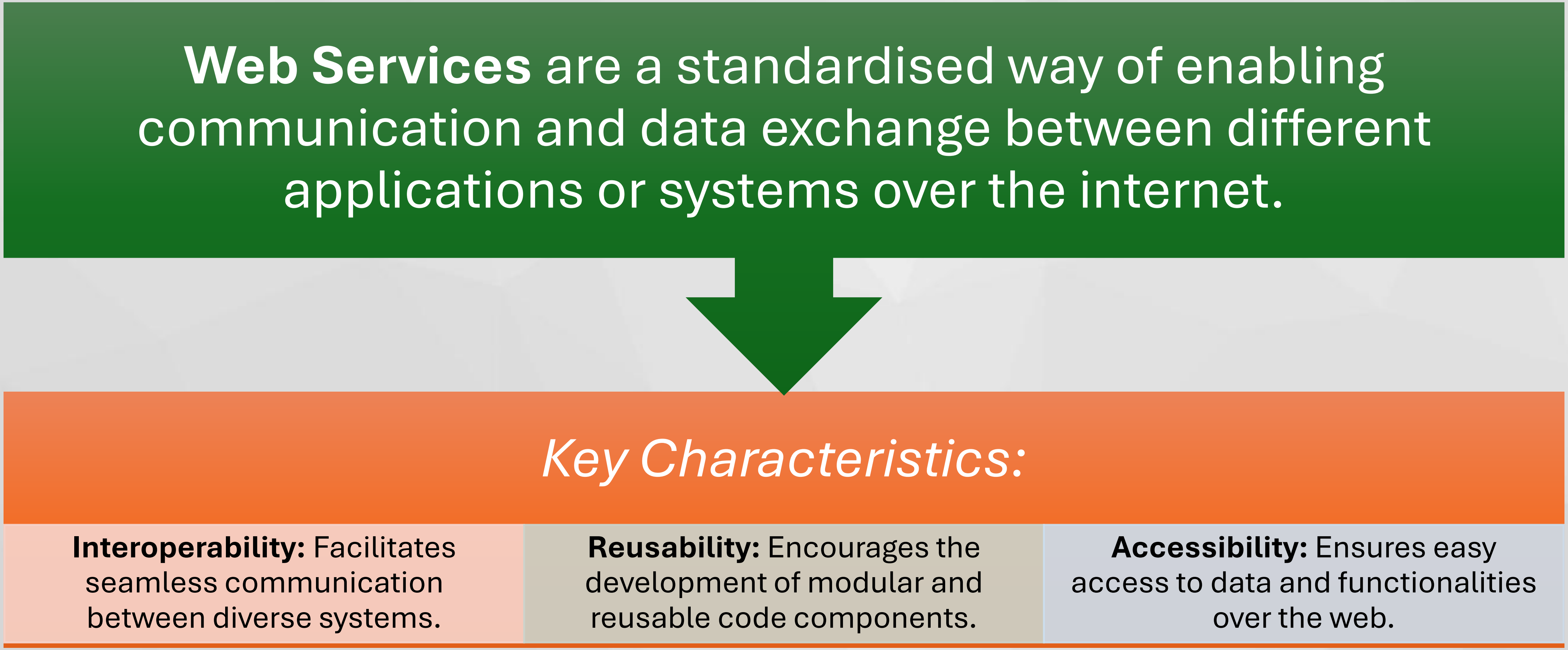
13/02/2025

Introduction

- *"Think about how different websites (banks, shipping companies, payment processors) interact when you buy something online. They 'talk' to each other behind the scenes to handle payment, shipping, and inventory. Web services enable this seamless communication.“*

- "How can different software systems, built with *different technologies* and running *on different platforms*, communicate and exchange data effectively?“
- "How can we build applications that are *modular*, *reusable*, and can *easily integrate* with other systems?“
- "How can we *expose* our application's *functionality* to other developers or businesses in a standardised way?"

What Are Web Services?



Web Services are a standardised way of enabling communication and data exchange between different applications or systems over the internet.

Key Characteristics:

Interoperability: Facilitates seamless communication between diverse systems.

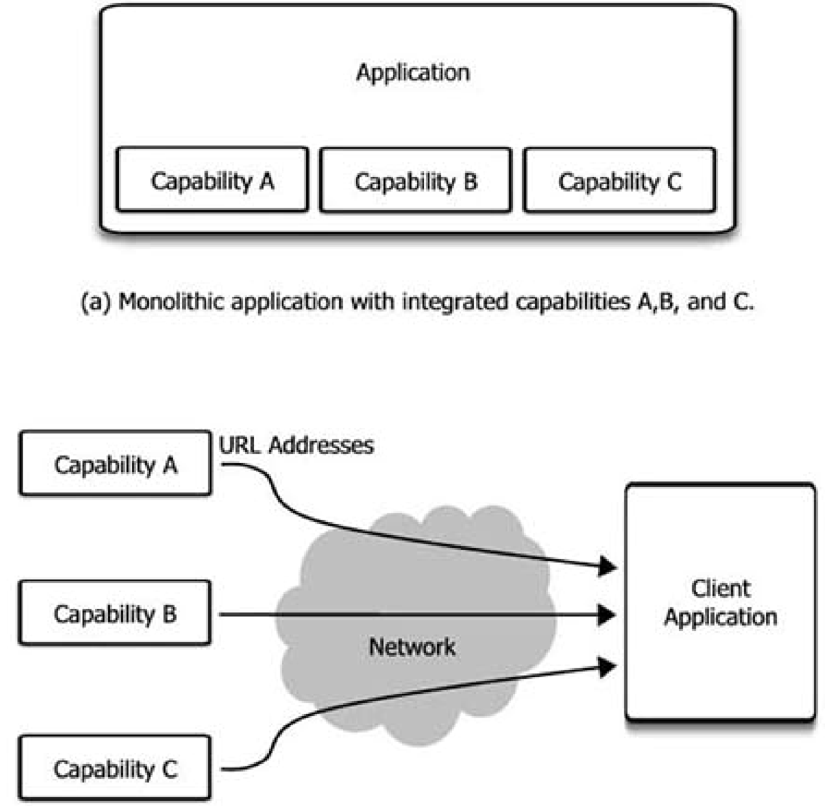
Reusability: Encourages the development of modular and reusable code components.

Accessibility: Ensures easy access to data and functionalities over the web.

What are web services?

Web services represent a new architectural paradigm for applications.

An application can use the capabilities of a Web service by simply invoking it across a network without having to integrate it.



(a) Monolithic application with integrated capabilities A,B, and C.

(b) Client application invoking remote Web services for capabilities A, B, and C.

Example: Online Store Payments

You want to let customers pay on your online store with their credit cards.

Traditional Approach:

- Work with banks and understand complex financial regulations.
- Build secure systems to handle sensitive credit card information.
- Implement fraud detection and security measures

Web Service Solution:

- You use a payment gateway web service (like Stripe, PayPal, or Braintree).
- When a customer wants to pay, your store redirects them to the payment gateway's secure page.
- The customer enters their card details on the payment gateway.
- The payment gateway handles the whole transaction and notifies you back on success or failure.

Web Services advantages

01

Dramatically cut application development costs

02

Reduce or eliminate many errors

03

Simplify application maintenance and customization

04

Significantly reduce time-to-market.

Characteristics of Web Services

Standardization

01

Serialisation Involves the use of established norms and protocols to ensure consistency and interoperability.

02

Use of standard protocols (e.g., SOAP, REST): leverage standardized communication protocols like SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) to enable seamless data exchange between applications.

03

Service Description: **WSDL (Web Services Description Language). OpenAPI (formerly Swagger)**

How Does Serialization Work?

Serialization Mechanism:

Your chosen programming language or library will provide a serialization mechanism. This could be a built-in function, a library, or a framework-specific serialization tool.

Format: The serialization mechanism determines the format for the byte stream:

XML: Human-readable text format, good for cross-platform data.

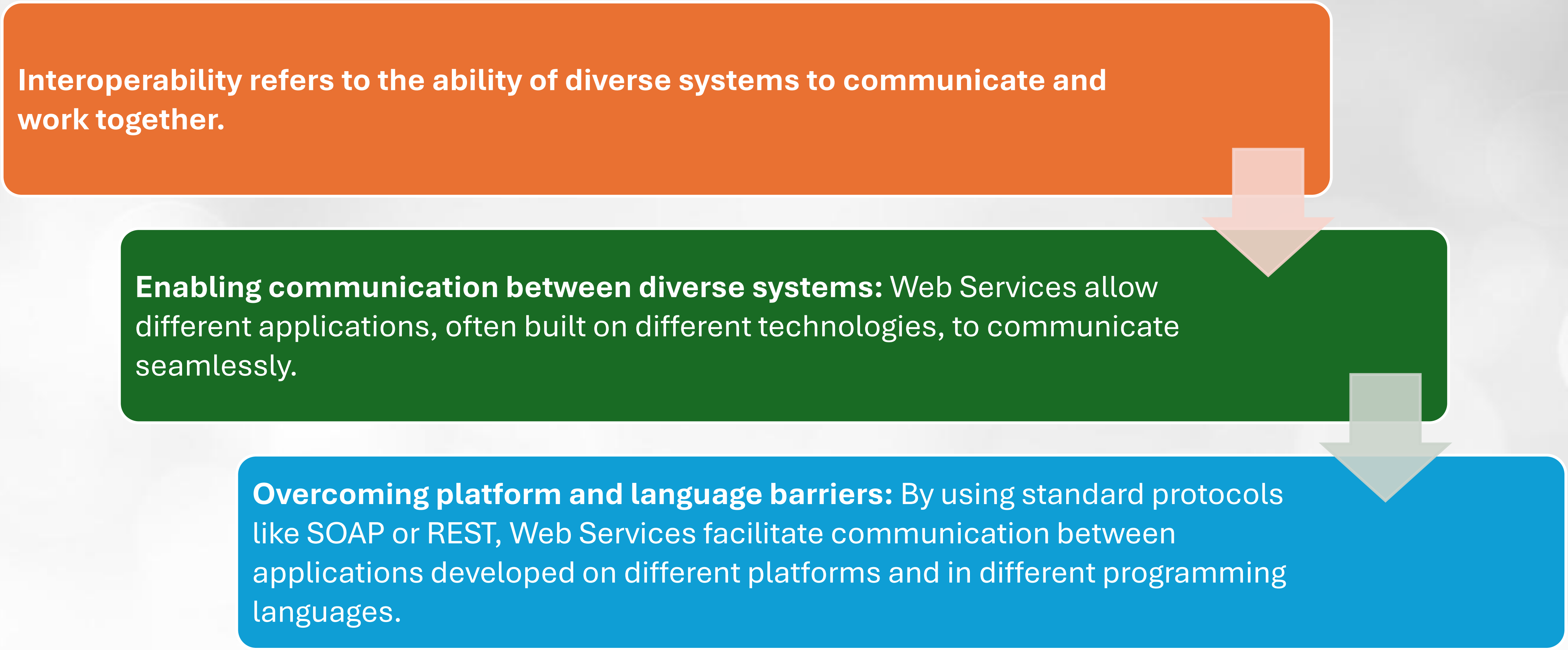
JSON: More lightweight text format, ideal for web interactions.

Binary formats: More compact, faster to process, but often platform-specific.

Standardization

- Imagine you're running a travel booking website and want to offer flight information from various airlines.
- Each airline might have its own API for providing flight data, with different formats and protocols.
- Question: What would be the problem?
- By using protocols like SOAP or REST, airlines can expose their data in a consistent way, making it easier for your website to understand and interpret the information.
- Additionally, standards for data formats like JSON or XML ensure the structure of the information is consistent, removing the need for complex parsing and adaptation.

Interoperability



Interoperability refers to the ability of diverse systems to communicate and work together.

Enabling communication between diverse systems: Web Services allow different applications, often built on different technologies, to communicate seamlessly.

Overcoming platform and language barriers: By using standard protocols like SOAP or REST, Web Services facilitate communication between applications developed on different platforms and in different programming languages.

Example - interoperability

E-commerce Order Processing

Imagine an online store that wants to streamline its order fulfillment process.

The Problem: The store's systems are a combination of older software and more recent additions:

- Its inventory management system runs on a Java platform.
- The shipping provider offers a .NET-based API for calculating rates and generating labels.
- The payment processing is handled by a third-party SaaS platform with a RESTful API.

The Goal: Make all these systems communicate seamlessly for automated order processing, regardless of their underlying technologies.

Discoverability

Discoverability refers to the ability to find and locate web services efficiently.

Methods for discovering and locating services: Various methods, such as service directories, registries, and standardized protocols, are employed to discover and locate web services within a network.

Importance of service directories and registries: Centralized repositories, like service directories and registries, play a crucial role in facilitating the discovery process. They provide a structured way for services to be cataloged and accessed.

Let's consider a scenario where discoverability plays a crucial role in web services

Imagine you're building a website that compares flights from various airlines to find the best deals for users.

Here's how discoverability of relevant web services is essential:

Finding Airline Web Services

Solutions for Discoverability?

Solutions for discoverability

UDDI (Universal Description, Discovery, and Integration):

UDDI registries act like directories or yellow pages for web services.

Airlines that want to make their flight search functionality available to external partners can register their services.

You could search these registries based on relevant keywords.

Web Search and Industry Repositories:

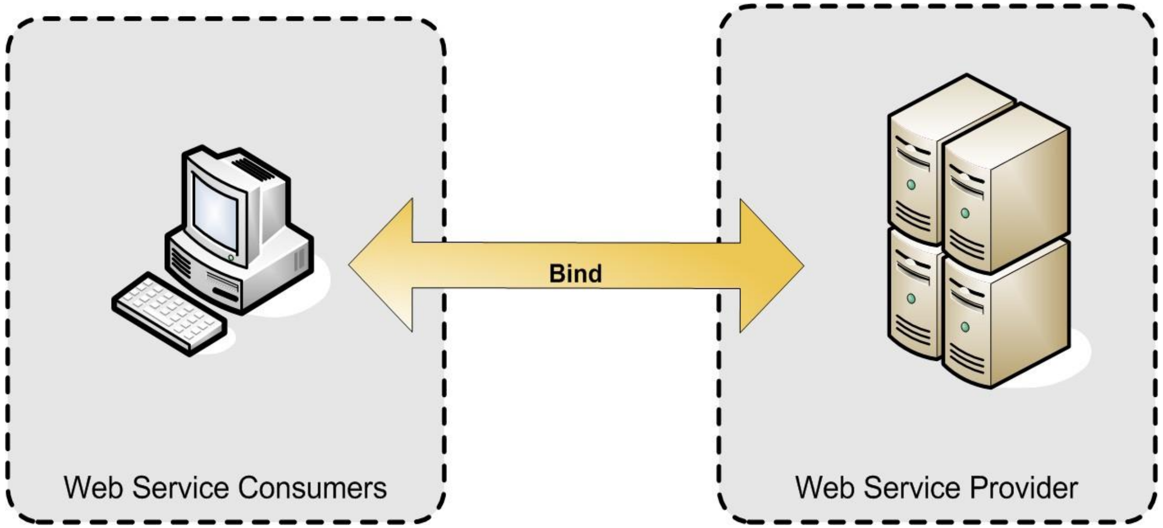
Airlines with web services might promote these on their developer portals or within industry-specific online repositories related to travel and booking.

The simplest Web service system has two participants:

A service producer (provider)

A service consumer (requester).

The provider presents the interface and implementation of the service, and the requester uses the Web service.

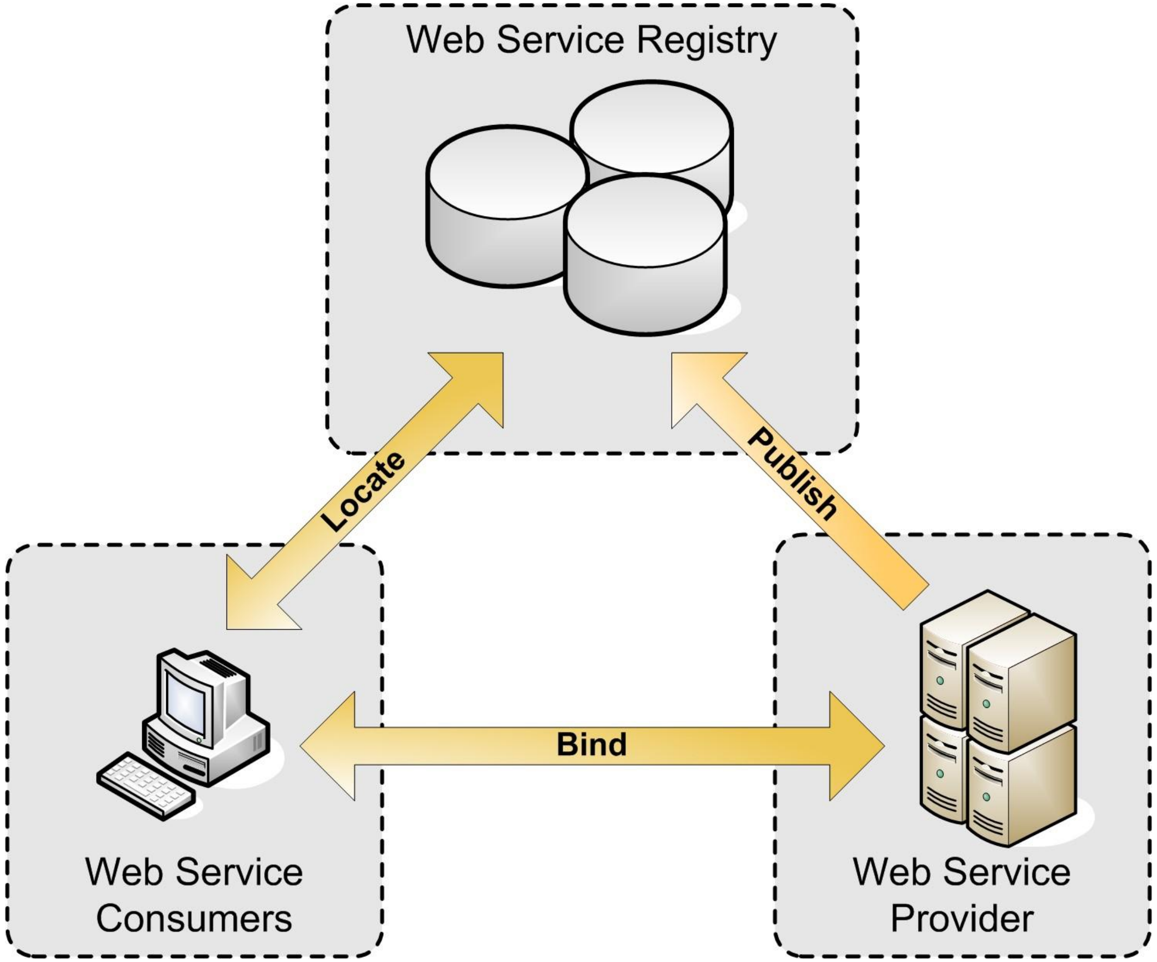


Web Service Architecture – Service Oriented

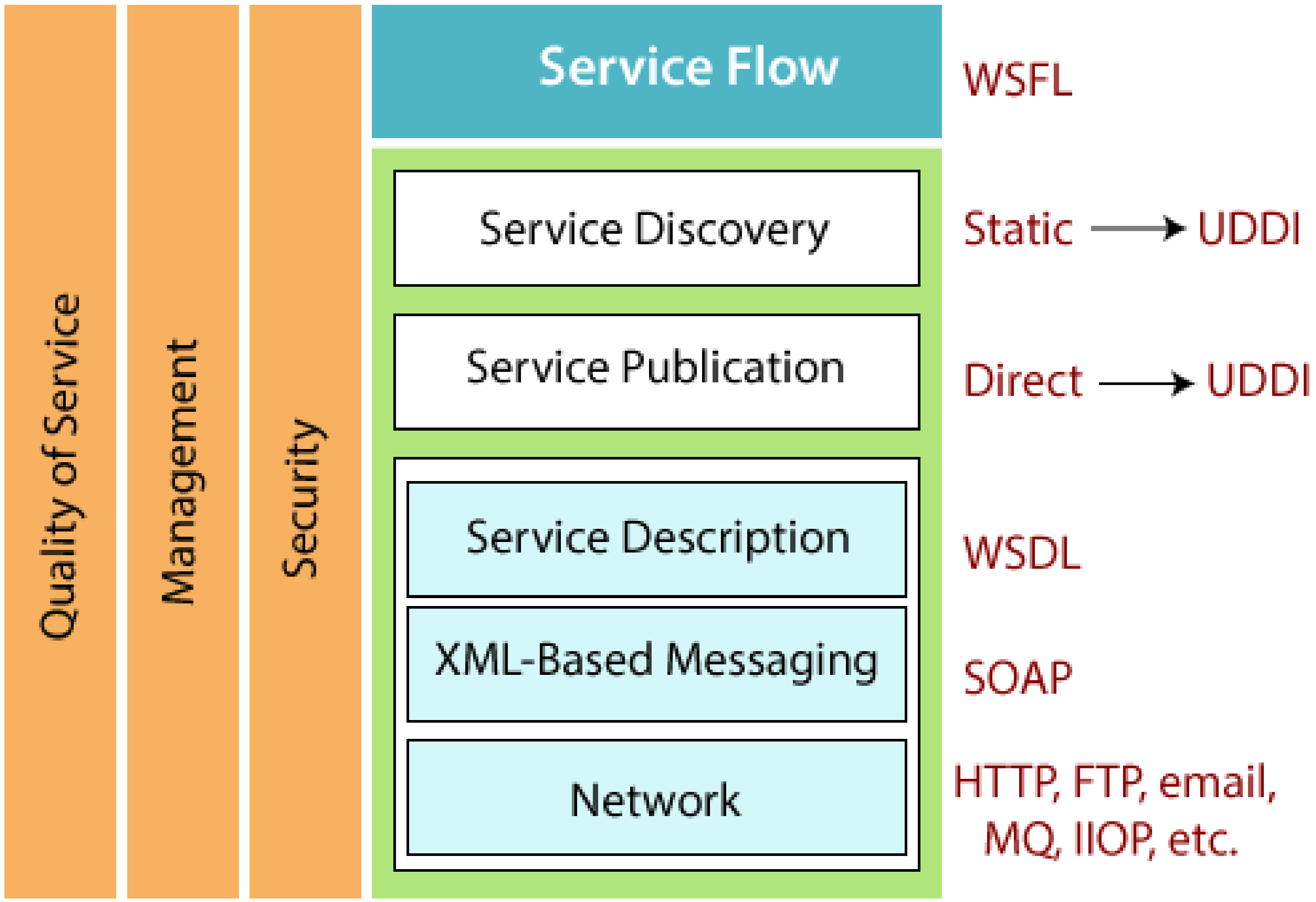
A registry, acts as a broker for Web services.

A provider, can publish services to the registry

A consumer, can then discover services in the registry



The Conceptual Web Services Stack



Quality of Service

Management

Security

Service Flow — WSFL

Service Discovery — Static → UDDI

Service Publication — Direct → UDDI

Service Description — WSDL

XML-Based Messaging — SOAP

Network — HTTP, FTP, email, MQ, IIOP, etc.

Quality of Service (QoS)

Reliability: Ensuring that services are available and function correctly, with minimal downtime and errors.

Performance: Optimizing response times, throughput, and resource utilization to deliver services efficiently.

Scalability: The ability of the system to handle increased load by adding resources or nodes.

Security: Ensuring the confidentiality, integrity, and authenticity of data and communication.

Best practices for reliability!

Availability:

- It should have minimal downtime or disruptions

Fault tolerance:

- recover from failures and continue operating without significant impact on its functionality or performance.

Data integrity:

- It should handle data validation, ensure proper data storage, and prevent data corruption or loss.

Monitoring and logging:

- track performance metrics, log errors and exceptions

Versioning and backward compatibility

Testing and validation

Documentation and support:significantly impacting

DISCOVERY: MECHANISMS FOR SERVICES TO DISCOVER EACH OTHER AND THEIR CAPABILITIES.

REGISTRATION: SERVICES NEED TO BE REGISTERED AND MAINTAIN UP-TO-DATE INFORMATION.

MONITORING: CONTINUOUS TRACKING OF SERVICE PERFORMANCE, AVAILABILITY, AND USAGE METRICS.

DEPLOYMENT: MANAGING THE DEPLOYMENT OF SERVICES IN VARIOUS ENVIRONMENTS.

SCALING: HANDLING THE SCALING OF SERVICES BASED ON DEMAND.

VERSIONING: MANAGING DIFFERENT VERSIONS OF SERVICES AND ENSURING COMPATIBILITY.

Web Standards (W3C)

Is an international community that develops standards and guidelines to ensure the long-term growth and accessibility of the World Wide Web

Responsible for developing and maintaining a wide range of technical specifications and guidelines that define the technologies used on the web.

Bringing together various stakeholders including industry leaders, researchers, developers, and public interest groups.

SOAP

SOAP (Simple Object Access Protocol) is a protocol for exchanging structured information in web services.

It is a messaging protocol that allows programs running on different operating systems to communicate with one another.

It is often employed in enterprise-level applications and scenarios where a strict and standardized communication protocol is required.

Protocol Specifications

Standards and Specifications

- SOAP is governed by industry standards and specifications.
- W3C (World Wide Web Consortium) defines the SOAP standard.

Versions

- SOAP 1.1 and SOAP 1.2 are the widely used versions.

HTTP and Other Transport Protocols

- SOAP messages can be transported over various protocols, including HTTP.

XML-Based Format for Message Structure

- **XML (eXtensible Markup Language)**
  - Platform-independent and human-readable.
  - Used to define the structure of SOAP messages.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:m="http://www.example.org">
  <soap:Header>
    <!-- Header content here -->
  </soap:Header>
  <soap:Body>
    <!-- Body content here -->
  </soap:Body>
</soap:Envelope>
```

SOAP

XML-based:

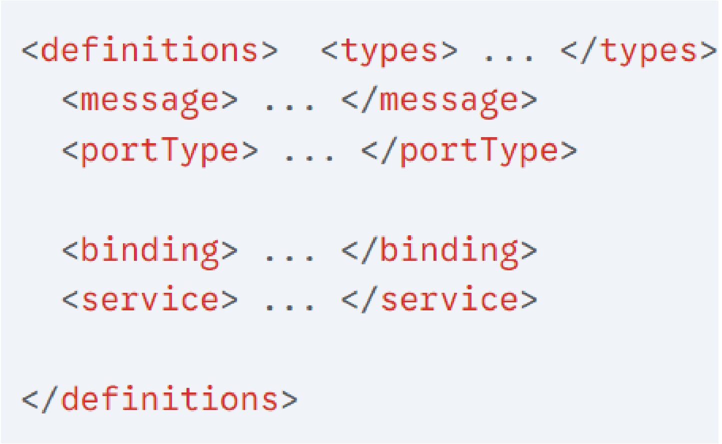
- SOAP messages are structured XML documents. This makes them platform-independent and human-readable (though slightly verbose).

Envelope Structure: SOAP messages have a specific structure. They always include an 'Envelope' element which contains a 'Header' (optional) and a 'Body'.

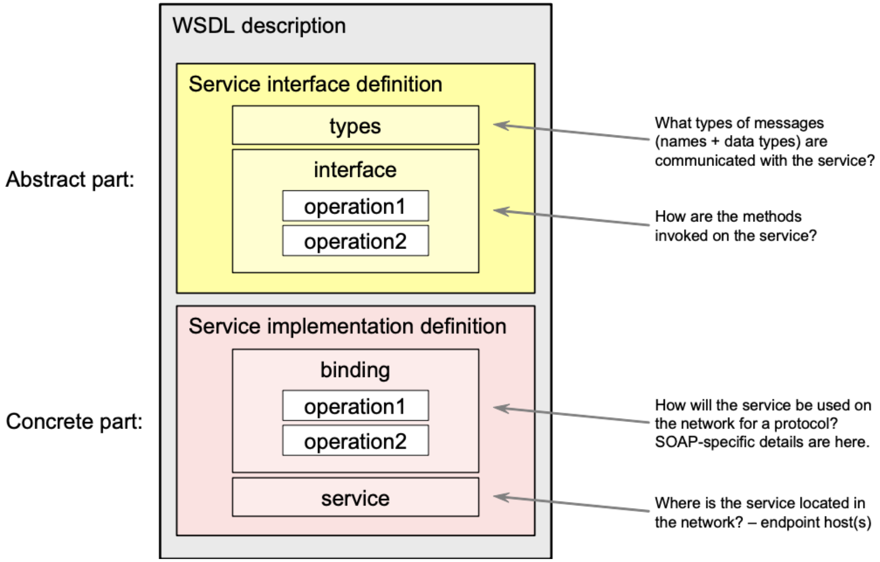
Header: Can carry extra information like security metadata, routing, or transaction details.

Body: Contains the actual message payload (e.g., information about a function call and its parameters).

WSDL



```
<definitions>  <types> ... </types>
  <message> ... </message>
  <portType> ... </portType>

  <binding> ... </binding>
  <service> ... </service>

</definitions>
```



What types of messages (names + data types) are communicated with the service?

How are the methods invoked on the service?

How will the service be used on the network for a protocol? SOAP-specific details are here.

Where is the service located in the network? – endpoint host(s)

Stands for: Web Services Description Language

Purpose: An XML-based language that describes the capabilities of a web service. Think of it as a 'contract' defining how a client application should interact with the web service.

WSDL

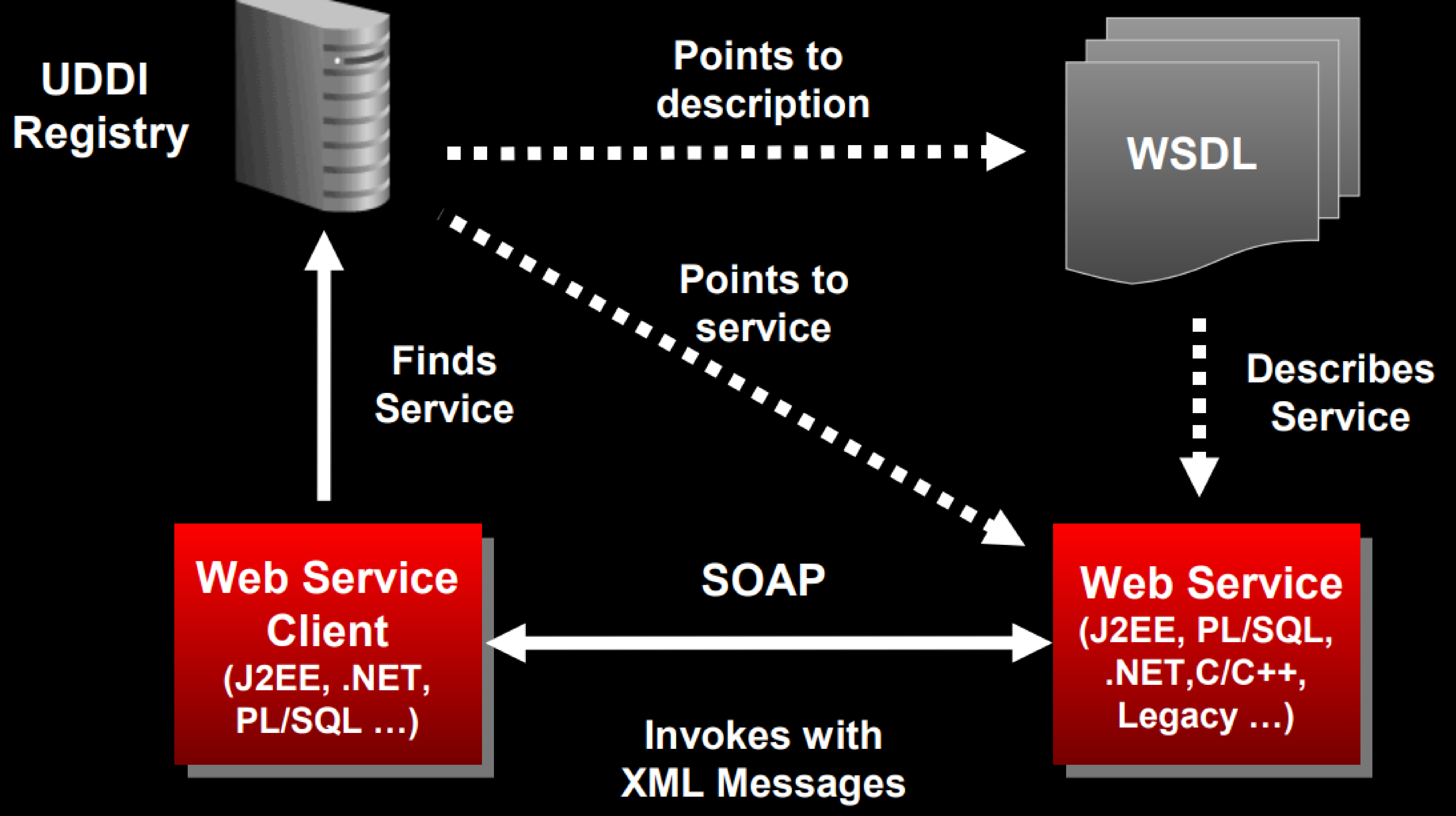
Operations: Lists the specific functions or actions the web service offers (e.g., calculateInterestRate, searchProducts).

Data Types: Describes the format of the input and output messages the web service expects and produces. This can be done using simple data types (string, integer) or more complex structures defined in XML Schema.

Binding: Specifies the communication protocol or mechanism (usually SOAP, but sometimes REST-like bindings are used) and the message format (e.g., how data is encoded within a SOAP message).

Endpoint: Provides the network address (URL) where the web service can be accessed.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:tns="http://www.example.com/calculator"
        targetNamespace="http://www.example.com/calculator">

  <message name="SubtractRequest">
   <part name="num1" type="xsd:int"/>
   <part name="num2" type="xsd:int"/>
  </message>

  <binding name="CalculatorBinding" type="tns:CalculatorPortType">
   <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="add">
    <soap:operation soapAction="http://www.example.com/calculator/add"/>
</operation>
   <operation name="subtract">
    <soap:operation
soapAction="http://www.example.com/calculator/subtract"/>
<output>
     <soap:body use="literal"/>
    </output>
   </operation>
  </binding>

  <service name="CalculatorService">
   <port name="CalculatorPort" binding="tns:CalculatorBinding">
    <soap:address location="http://www.example.com/calculator/service"/>
   </port>
  </service>
</definitions>
```



UDDI Registry

Points to description

WSDL

Points to service

Finds Service

Describes Service

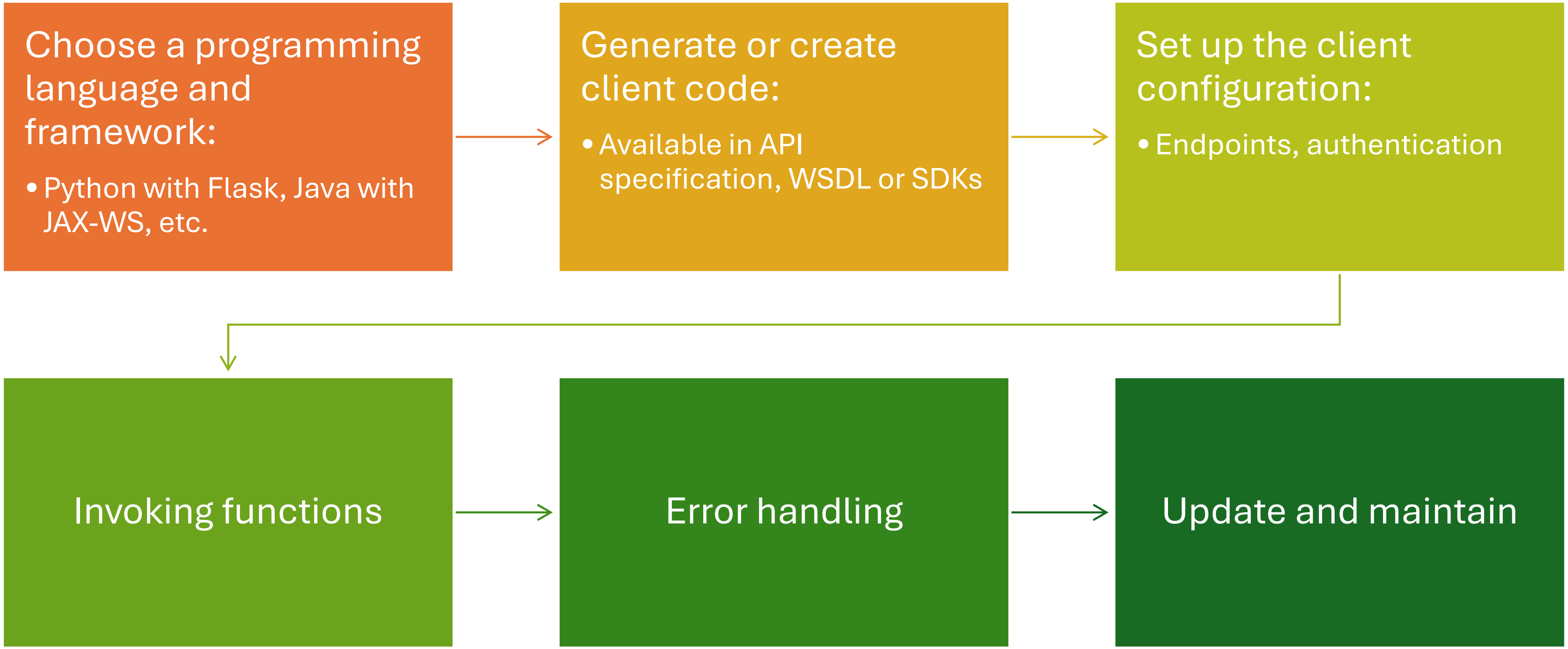
Web Service Client (J2EE, .NET, PL/SQL …)

SOAP

Web Service (J2EE, PL/SQL, .NET,C/C++, Legacy …)

Invokes with XML Messages

The implementation and design of a web service client



Choose a programming language and framework:

- Python with Flask, Java with JAX-WS, etc.

Generate or create client code:

- Available in API specification, WSDL or SDKs

Set up the client configuration:

- Endpoints, authentication

Invoking functions

Error handling

Update and maintain

RESTful Web Services

REST (Representational State Transfer) is an architectural style for designing networked applications.

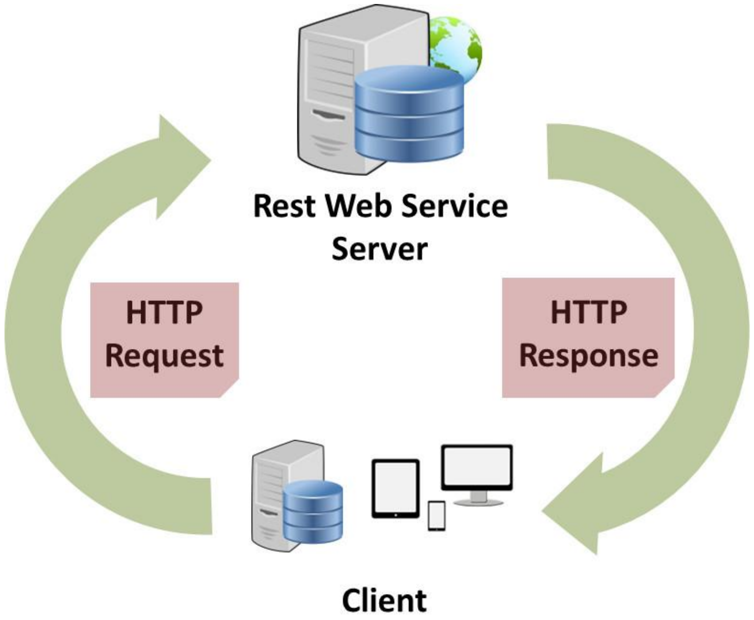
RESTful Web Services use the principles of REST, emphasizing simplicity, scalability, and statelessness.

REST is based on principles such as stateless communication, resource identification, and uniform interfaces, contributing to its popularity for web service development.

Introduction to RESTful Services

Representational State Transfer (REST) is an architectural style for designing networked applications.

It’s not a protocol or a standard, but a set of guiding principles and constraints.



Principles of REST - Stateless Communication

In REST, statelessness refers to when the client is responsible for storing and handling the session-related information on its own side.

Each request from the client to the server must contain all the information needed to process that request.

Principles of REST - Client-Server Architecture

In a RESTful architecture, the server and the client are clearly isolated from each other.

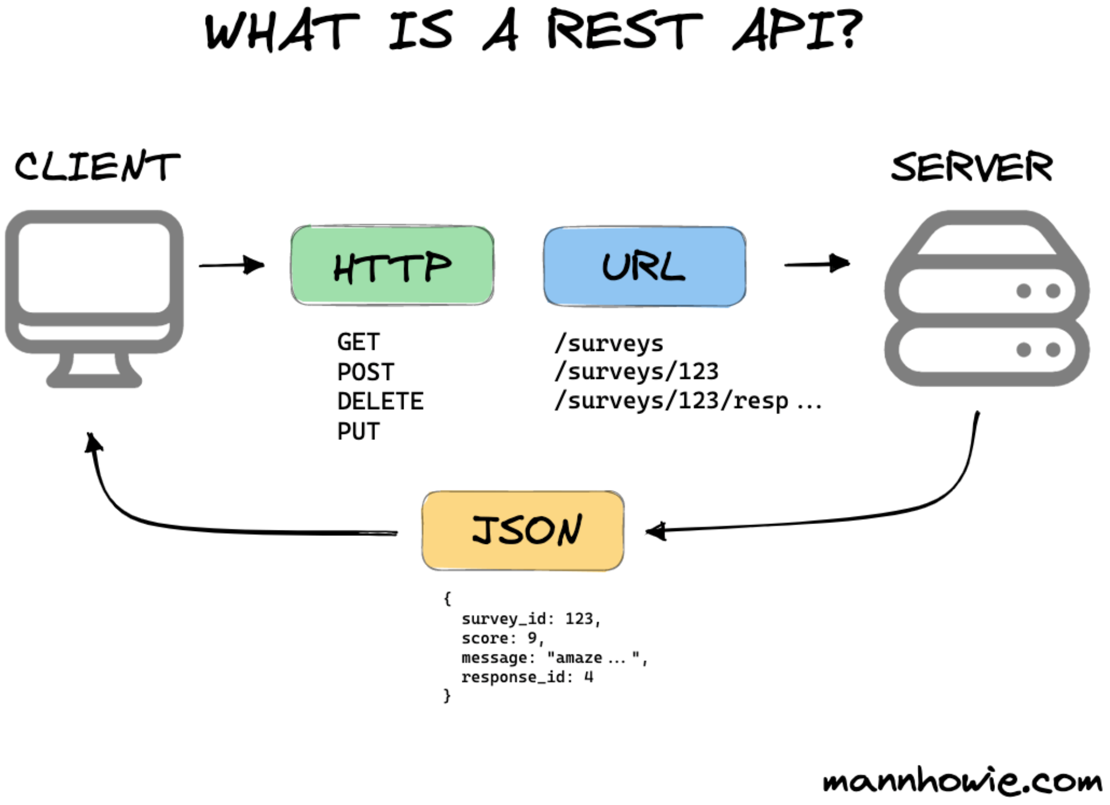
While the server doesn’t know the user interface, the client doesn’t know the application’s business logic or how the application persists data.

Principles of REST - Uniform Interface

REST defines a consistent and uniform interface for interactions between clients and servers.

For example, the HTTP-based REST APIs make use of the standard HTTP methods (GET, POST, PUT, DELETE, etc.) and the URIs (Uniform Resource Identifiers) to identify resources.

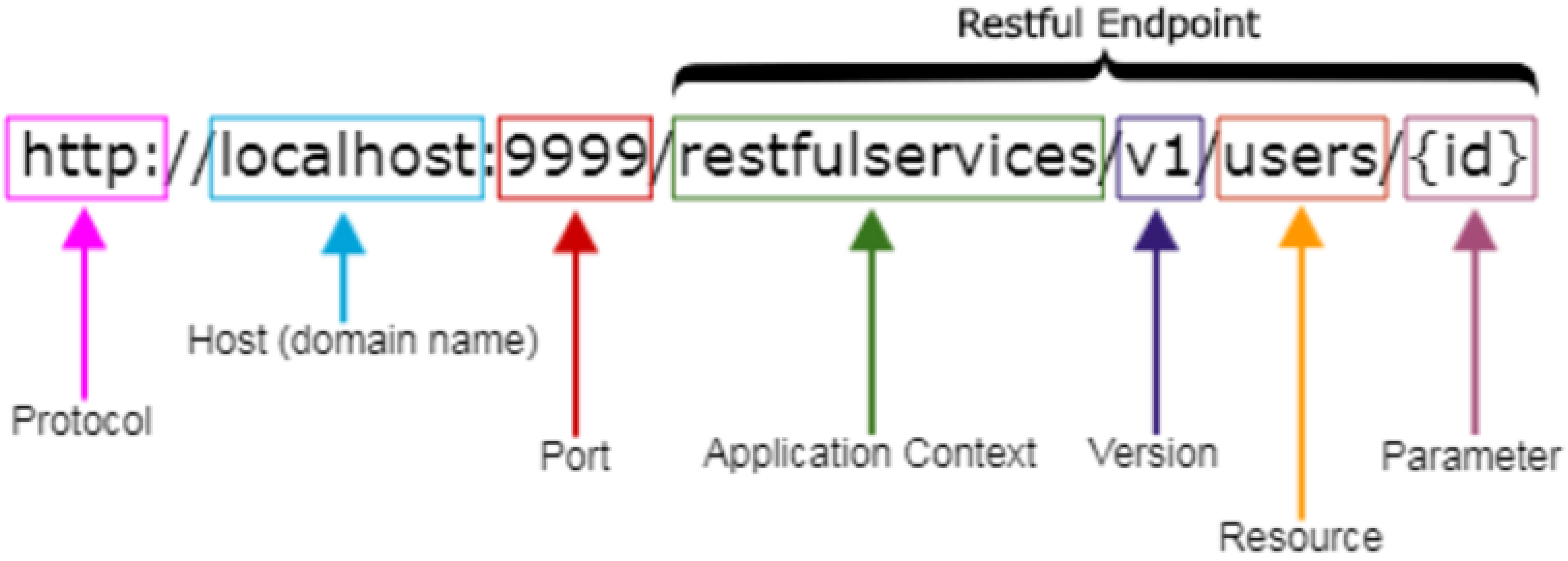
Principles of REST - Uniform Interface



GET
POST
DELETE
PUT

/surveys
/surveys/123
/surveys/123/resp ...

```
{
  survey_id: 123,
  score: 9,
  message: "amaze ... ",
  response_id: 4
}
```



Restful Endpoint

http://localhost:9999/restfulservices/v1/users/{id}

Protocol

Host (domain name)

Port

Application Context

Version

Resource

Parameter

Using an external service's functionality in your client web service

Obtain API credentials:

Many services require API keys or access tokens

Read the documentation:

how to make requests, what endpoints are available, and what data formats are expected.

Make HTTP requests:

appropriate HTTP methods (GET, POST, PUT, DELETE) to send requests to the endpoints

Handle the responses:

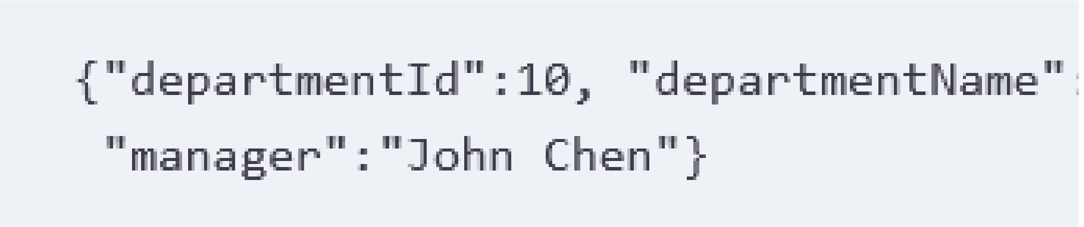
responses may be in various formats such as JSON, XML, or others

JSON

Lightweight data interchange format.

Enclosed in curly braces {}.

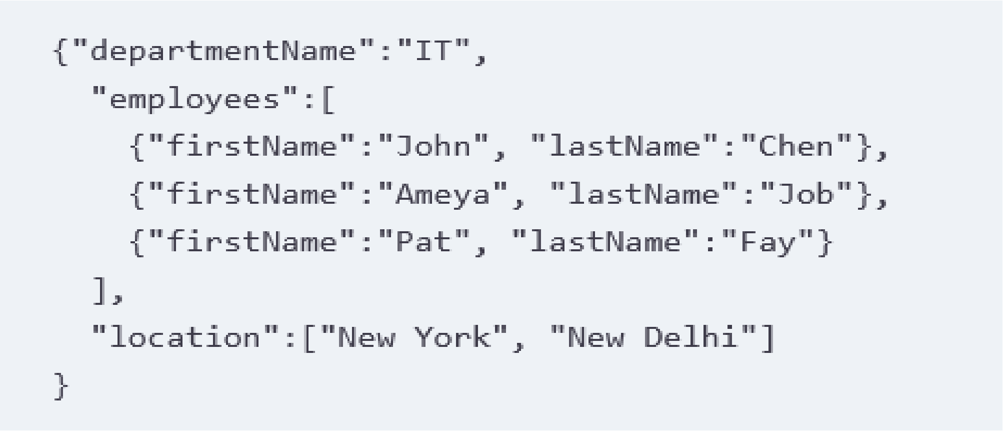
Consists of key-value pairs.



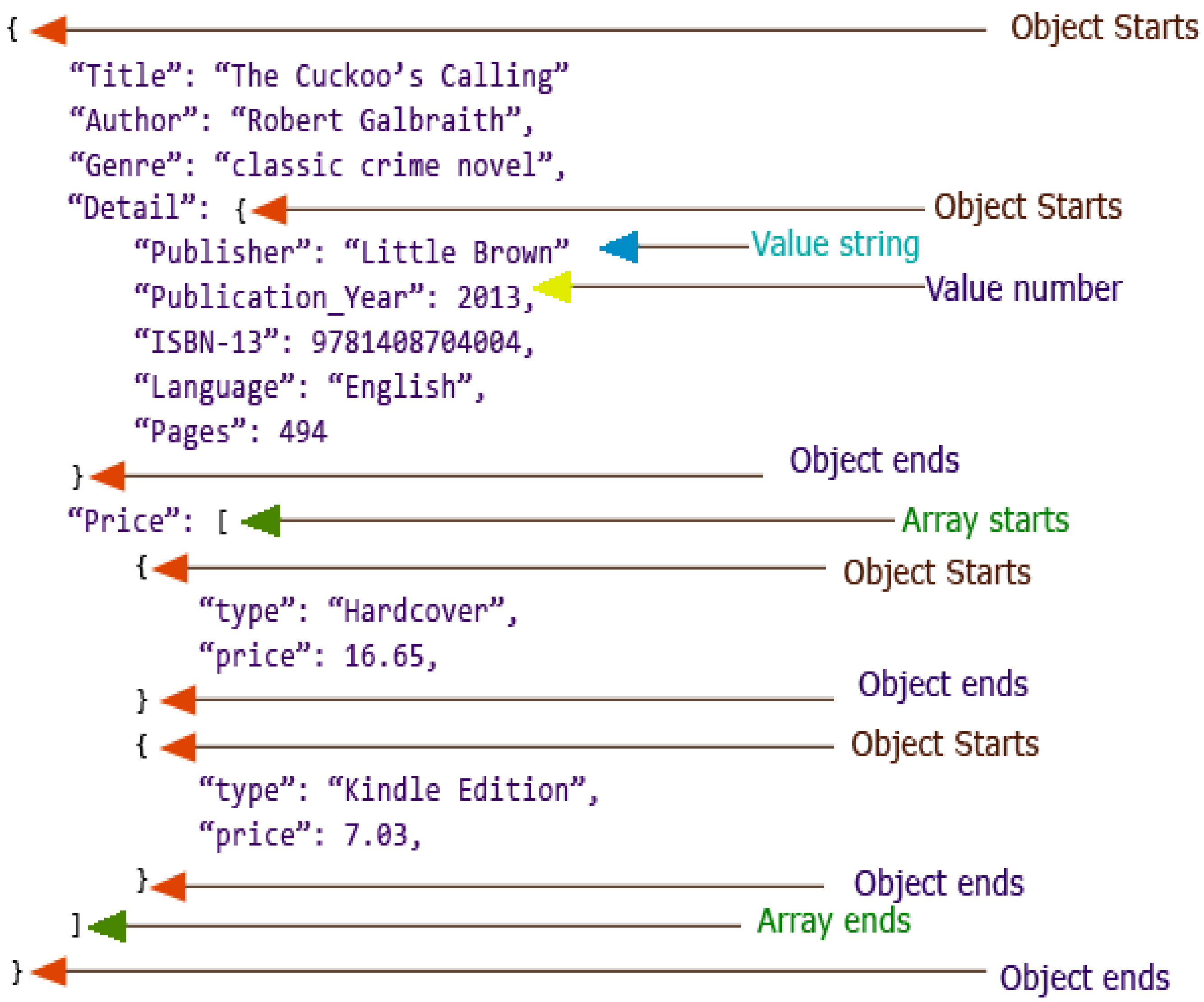
```
{"departmentId":10, "departmentName"
 "manager":"John Chen"}
```

JSON data syntax

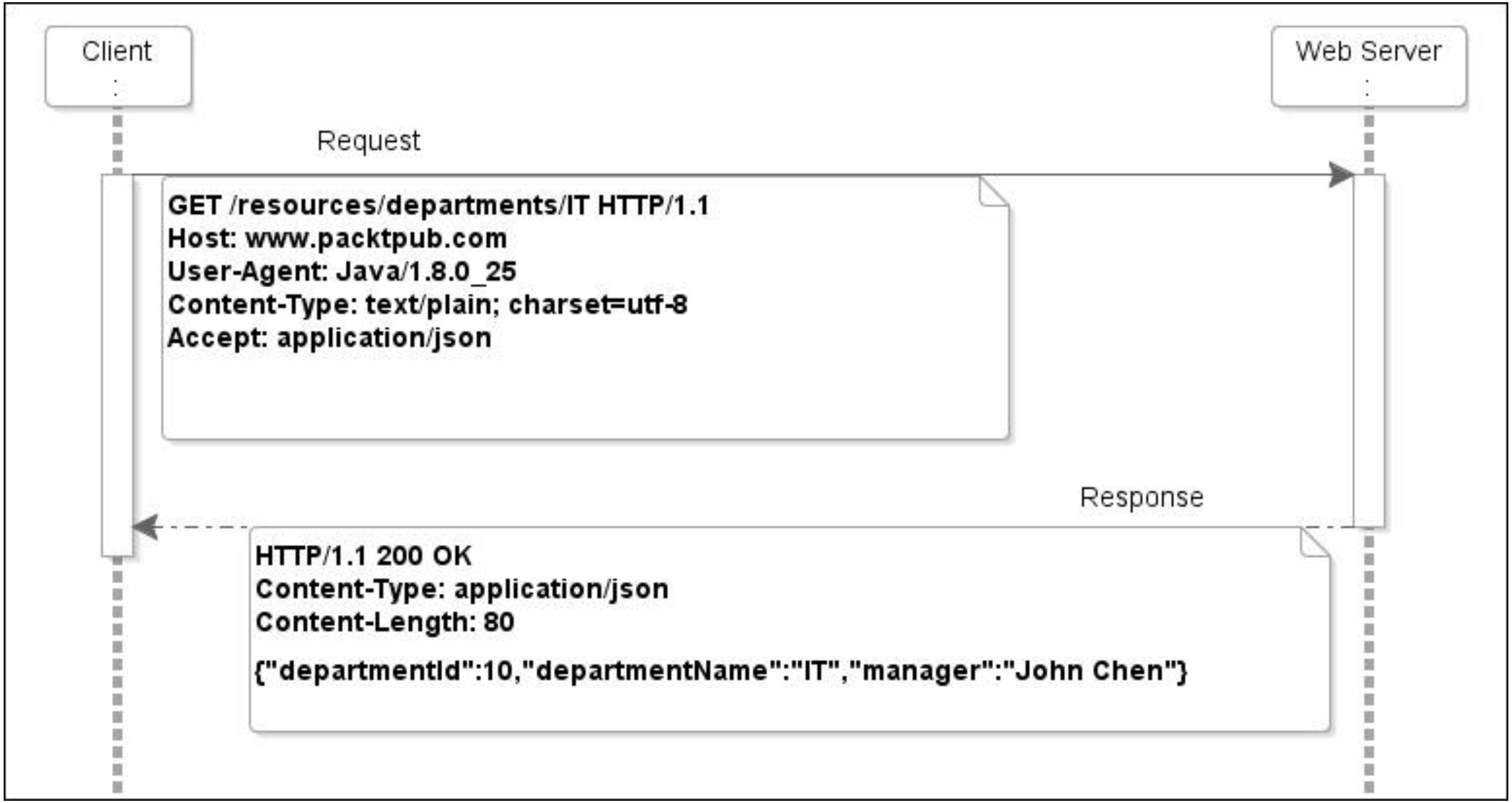
- Arrays are enclosed in square brackets ([ ]), and their values are separated by a comma (,).
- Each value in an array may be of a different type, including another array or an object.



```
{"departmentName":"IT",
  "employees":[
    {"firstName":"John", "lastName":"Chen"},
    {"firstName":"Ameya", "lastName":"Job"},
    {"firstName":"Pat", "lastName":"Fay"}
  ],
  "location":["New York", "New Delhi"]
}
```

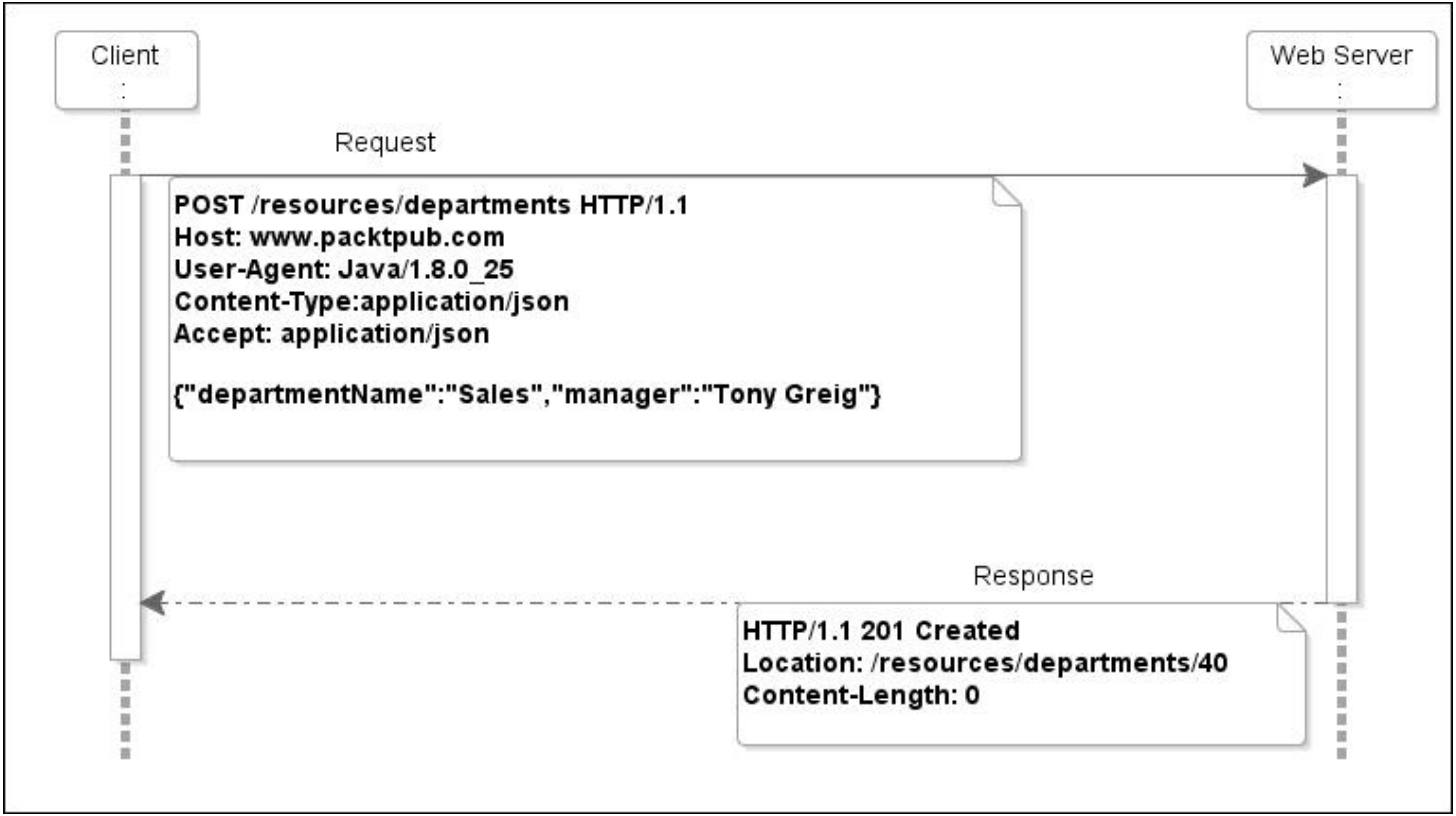


```
{
   “Title”: “The Cuckoo’s Calling”
   “Author”: “Robert Galbraith”,
   “Genre”: “classic crime novel”,
   “Detail”: {
       “Publisher”: “Little Brown”
       “Publication_Year”: 2013,
       “ISBN-13”: 9781408704004,
       “Language”: “English”,
       “Pages”: 494
   }
   “Price”: [
       {
           “type”: “Hardcover”,
           “price”: 16.65,
       }
       {
           “type”: “Kindle Edition”,
           “price”: 7.03,
       }
   ]
}
```



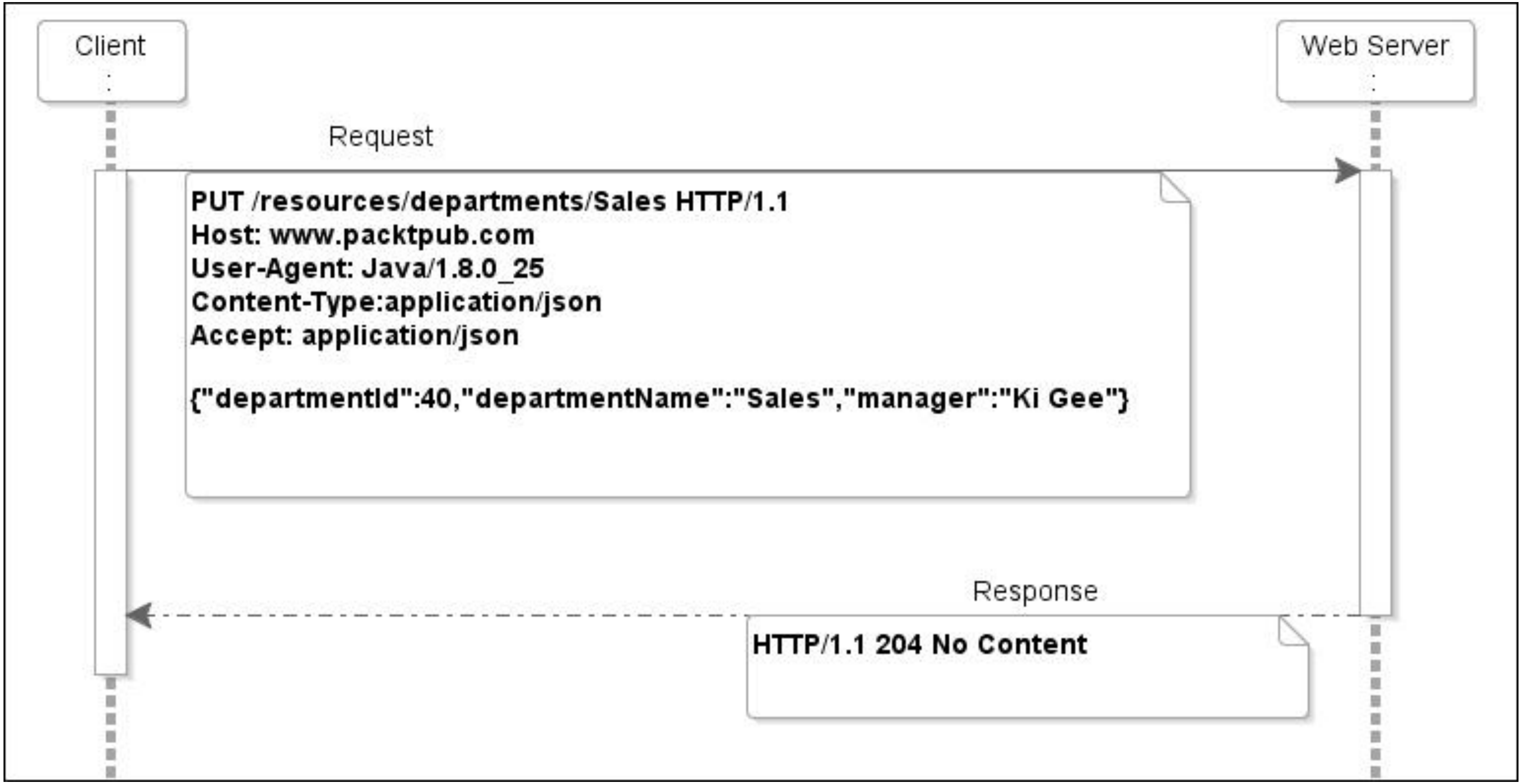
```
GET /resources/departments/IT HTTP/1.1
Host: www.packtpub.com
User-Agent: Java/1.8.0_25
Content-Type: text/plain; charset=utf-8
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 80

{"departmentId":10,"departmentName":"IT","manager":"John Chen"}
```



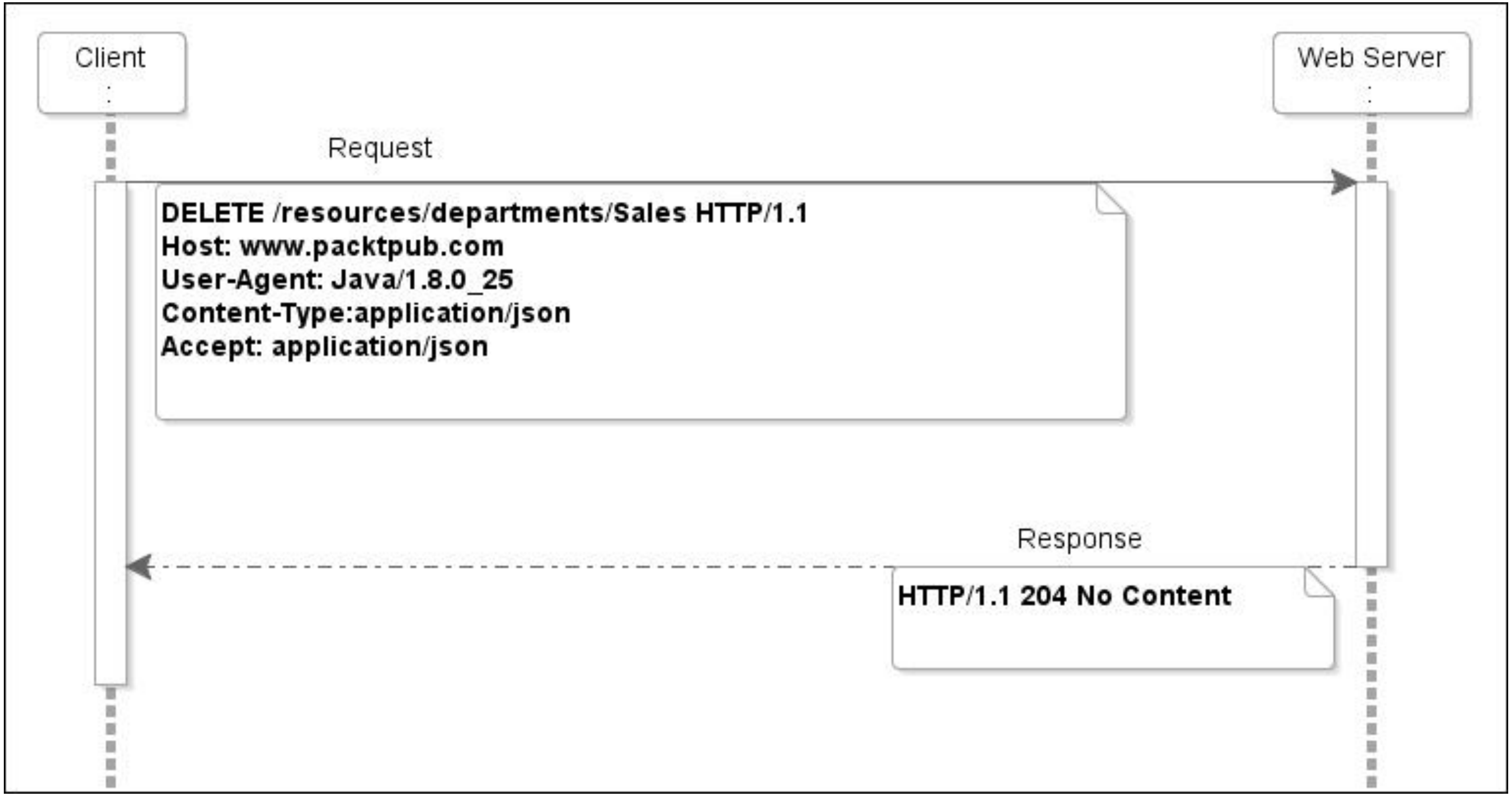
```
POST /resources/departments HTTP/1.1
Host: www.packtpub.com
User-Agent: Java/1.8.0_25
Content-Type:application/json
Accept: application/json

{"departmentName":"Sales","manager":"Tony Greig"}
```

```
HTTP/1.1 201 Created
Location: /resources/departments/40
Content-Length: 0
```



```
PUT /resources/departments/Sales HTTP/1.1
Host: www.packtpub.com
User-Agent: Java/1.8.0_25
Content-Type:application/json
Accept: application/json

{"departmentId":40,"departmentName":"Sales","manager":"Ki Gee"}
```

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



```
DELETE /resources/departments/Sales HTTP/1.1
Host: www.packtpub.com
User-Agent: Java/1.8.0_25
Content-Type:application/json
Accept: application/json
```

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

The Travel Agency App

What is a monolith: It's like a giant castle where all the parts of your app (booking flights, reserving hotels, managing user accounts) live together inside the same walls.

Pros: Can be simpler to start with.

Cons: As your app grows, it can be hard to change or scale one part without affecting everything else.

Designing the Web Service

What does our travel app need to do? Let's list the features:

- Search for flights (destination, dates, etc.)
- View flight options and prices
- Book flights
- Search for hotels
- Book hotels
- Manage user accounts (login, profiles, trip history)

What kind of information the app needs from outside to work?

Flights: Needs data from airline systems (schedules, prices)

Hotels: Needs data from hotel booking services

User accounts: Our app will likely store this data itself

Web service endpoints

Each feature will have its own 'endpoint'. An endpoint is like a special web address your app uses to make a request:

/search-flights

/book-flight

/search-hotels

and so on...