



Website Design

CSc 47300

Week 3

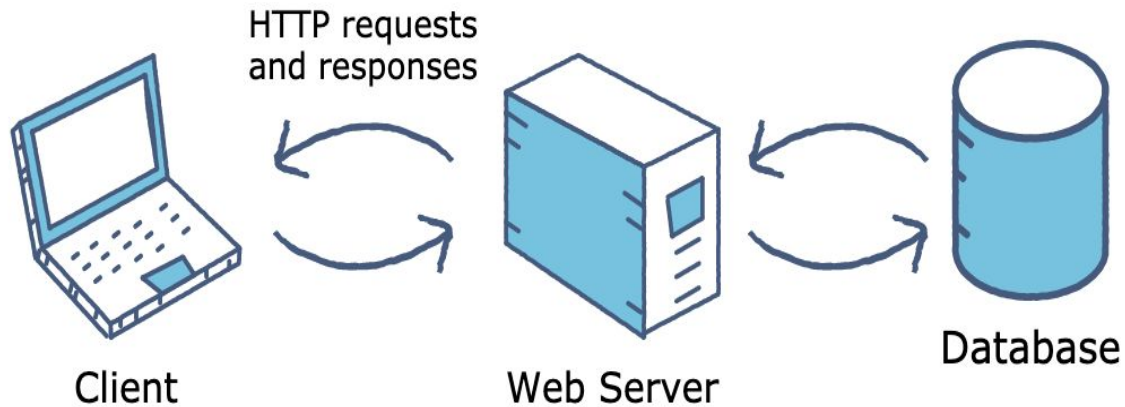


Overview

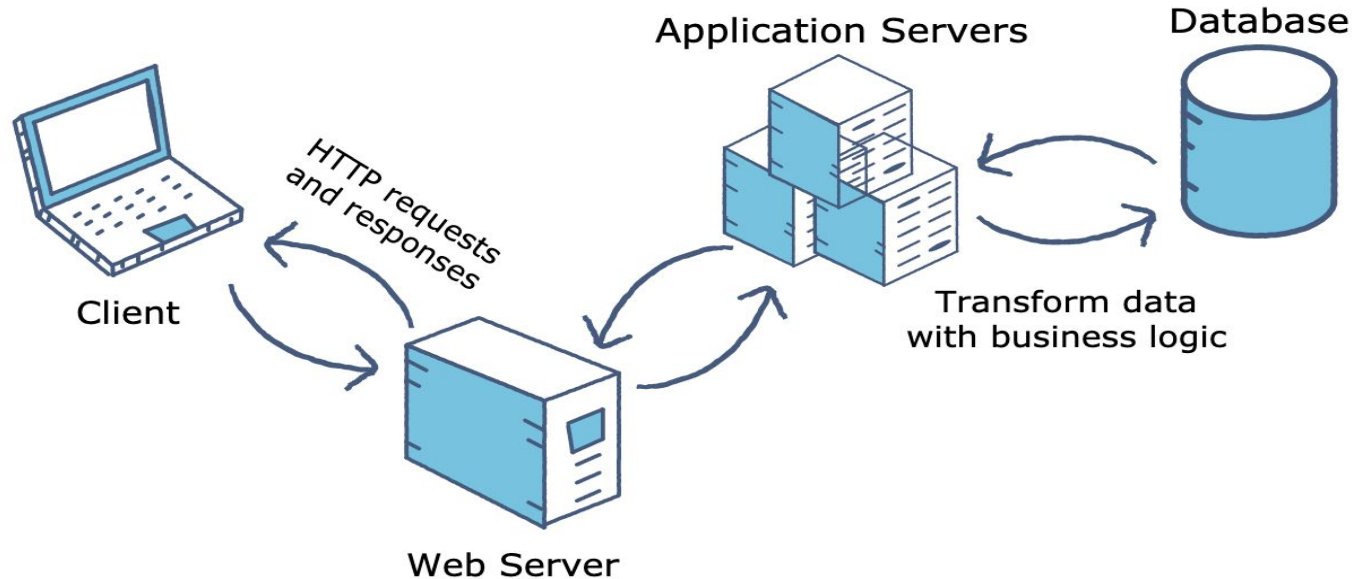
- Announcements
- Web Applications Architecture
- Web Server
- Application Server
- APIs
- Flask
- Design Exercise

**Small tools which do one thing,
and do it well.**

Web Application Architecture



Web Application Architecture (cont.)



Web Server



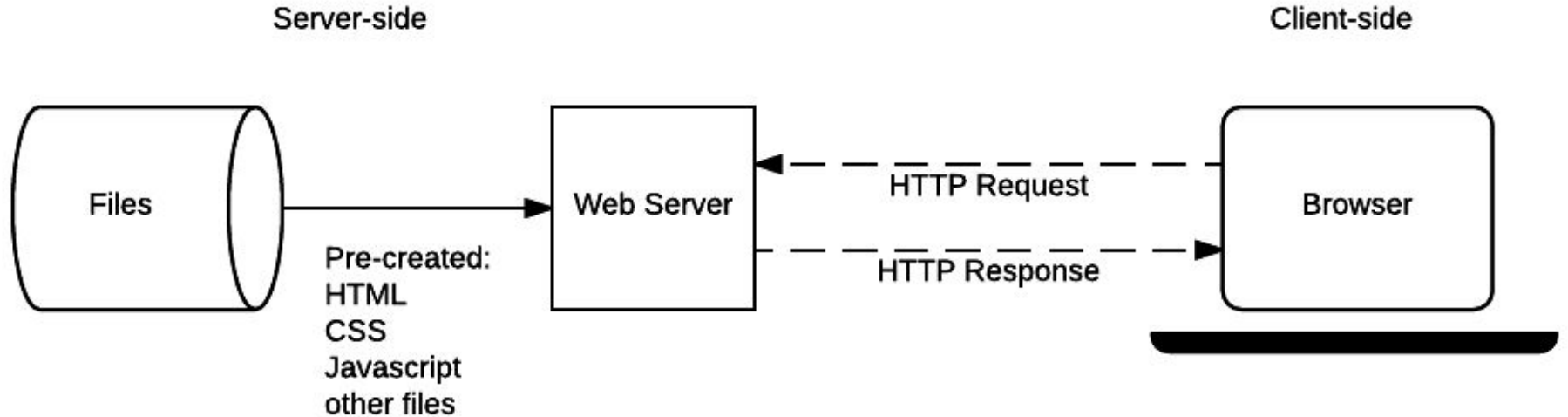
What is a Web Server?

A **web server** stores and delivers the content for a website – such as text, images, video, and application data – to clients that request it.

The most common type of client is a web browser program, which requests data from your website when a user clicks on a link or downloads a document on a page displayed in the browser.

At the most basic level, whenever a browser needs a file that is hosted on a web server, the browser requests the file via HTTP. When the request reaches the correct web server, the HTTP server accepts the request, finds the requested document, and sends it back to the browser, also through HTTP.

What is a Web Server? (cont.)





Popular Web Servers



Application Server

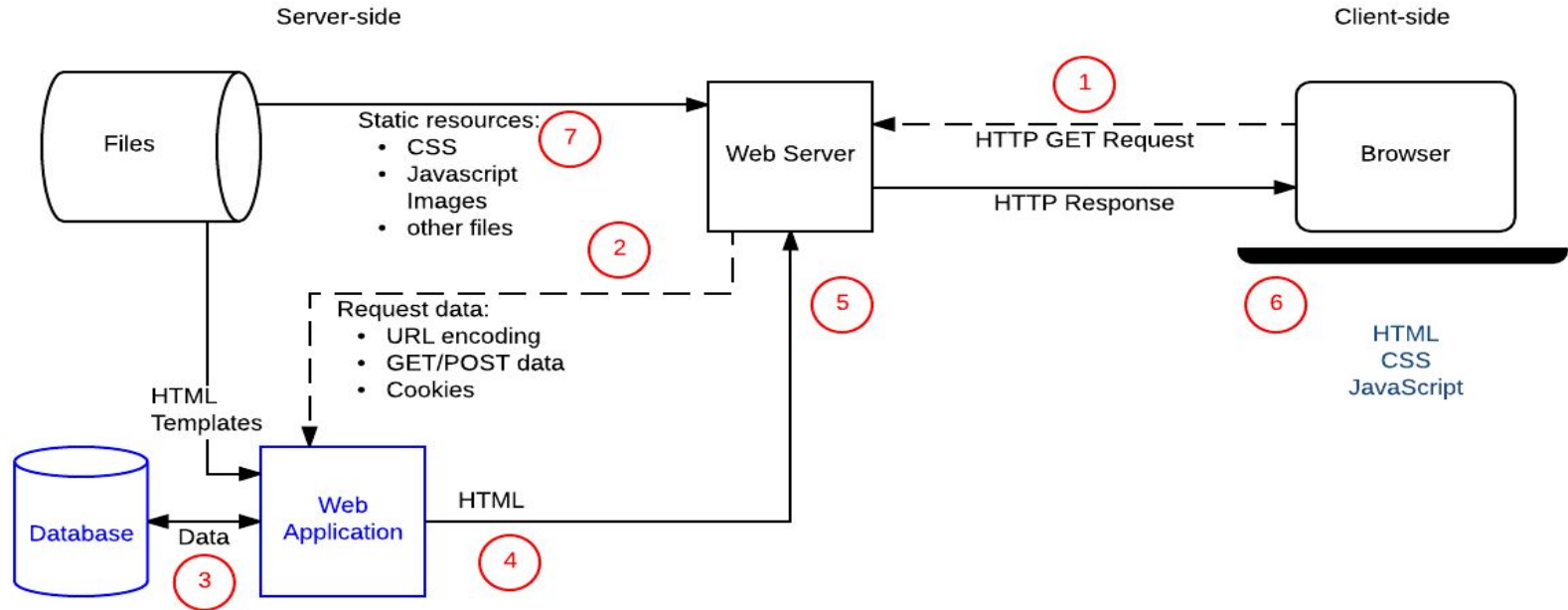


What is an Application Server?

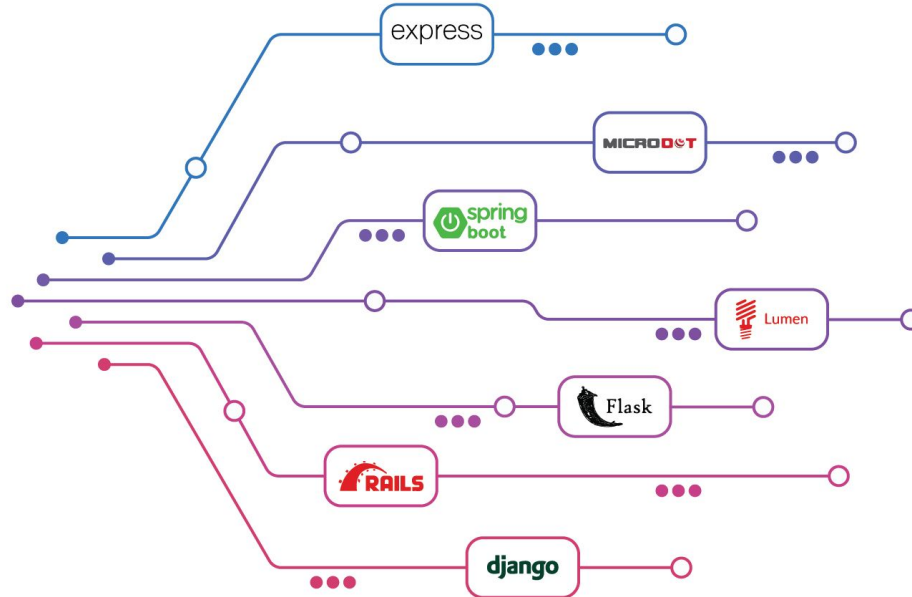
An **application server** typically can deliver web content too, but its primary job is to enable interaction between end-user clients and server-side application code—the code representing what is often called business logic—to generate and deliver dynamic content, such as transaction results, decision support, or real-time analytics.


The client for an application server can be the application's own end-user UI, a web browser, or a mobile app, and the client-server interaction can occur via any number of communication protocols.

What is an Application Server? (cont.)



Popular Web Application Frameworks

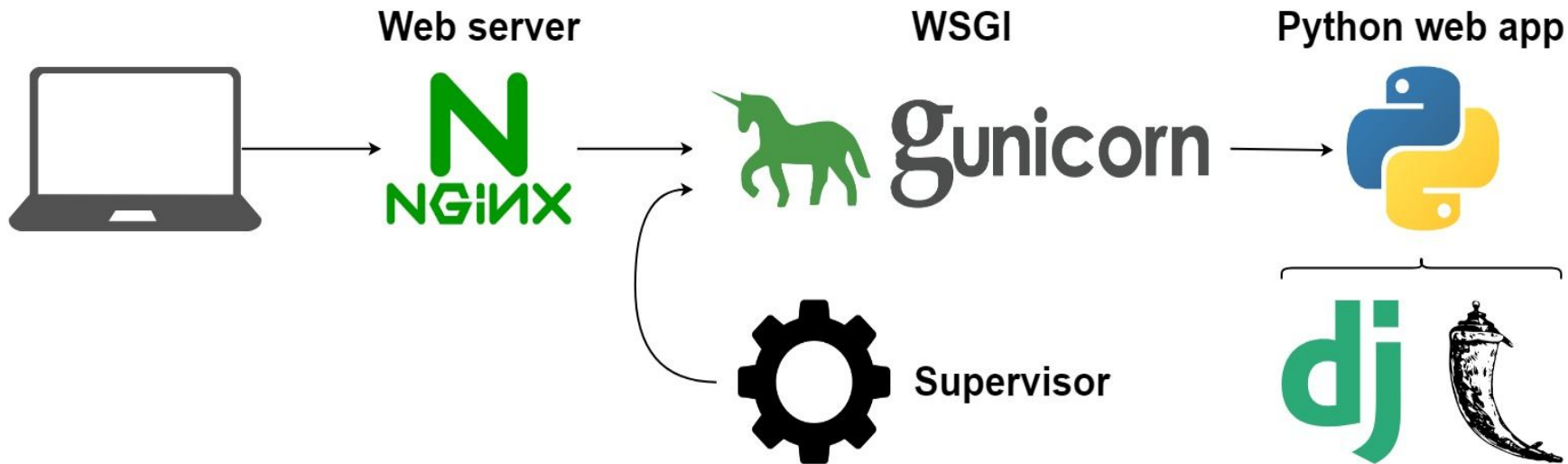




How Do Application Servers and Web Servers Work Together?

In a typical deployment, a website that provides both static and dynamically generated content runs web servers for the static content and application servers to generate content dynamically.

Bird's-Eye View



API

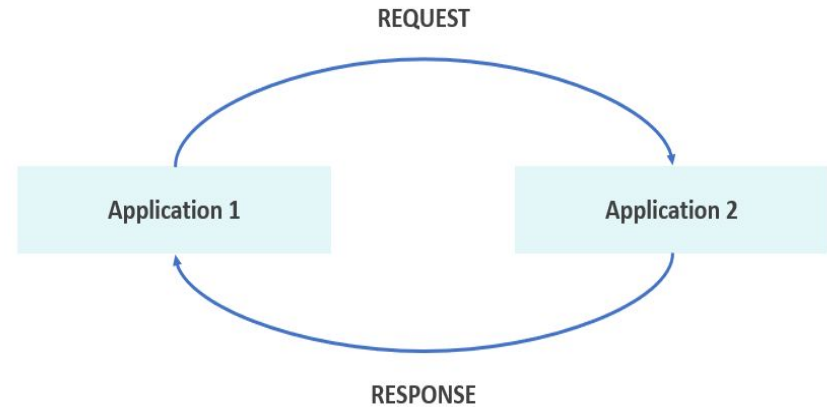


What is an API?

API stands for **Application Programming Interface**. It is basically how different machines and software talk to each other to create ever more complex applications. Even though APIs can simply refer to an interface to a local library, or the libraries themselves. For purposes of these slides, APIs are assumed to be over the network to connect clients and servers.

What is an API? (cont.)

- The communication between the applications cannot happen unless one of them initiates it, just like in real life.
- Thus, one of the parties involved needs to send a Request to the other one. The latter will return a Response, after running a Program.





Why build APIs?

- These days, very few applications are stand alone.
 - If you build a mobile app, most likely you'll need a build a server to provide resources for that mobile app.
 - You'll have to build internal APIs. And you'll consume many third party APIs to avoid reinventing the wheel.
- Other developers and companies may want to interface with your APIs and leverage the technology or data.
 - Your API is no longer just a tool to power your mobile app, but a product and a platform in it's own right.
- An API can act as a platform for new partnership opportunities, new revenue channels, or even new features for your organization.
 - Salesforce generates 50% of its revenue through their APIs, and Expedia generates over 90% of its revenue through their APIs.



Terminology

API → An API (Application Programming Interface) is an interface between two applications that enables them to communicate with each other. Often times, it refers to RESTful APIs over HTTP with JSON, as it is most common.

Client → The software that are consumers of the APIs.

Services → This refers to the the software that resides on the servers that provides the API to clients.

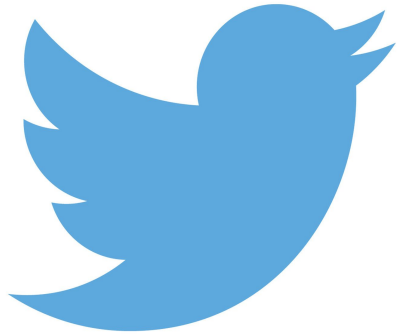


Example APIs





Example APIs



stripe



APIs in Play

- When you search for a flight in an application like Expedia, the web app returns all the available flights to the indicated destination.
 - It does this by talking to other applications, which are the websites of the various airlines.
- When you search for a hotel on Booking, you can retrieve data regarding the location of hotels, the availability of rooms in each of them and the services offered.
 - It does this by talking to other applications, which are the websites of the various hotels.

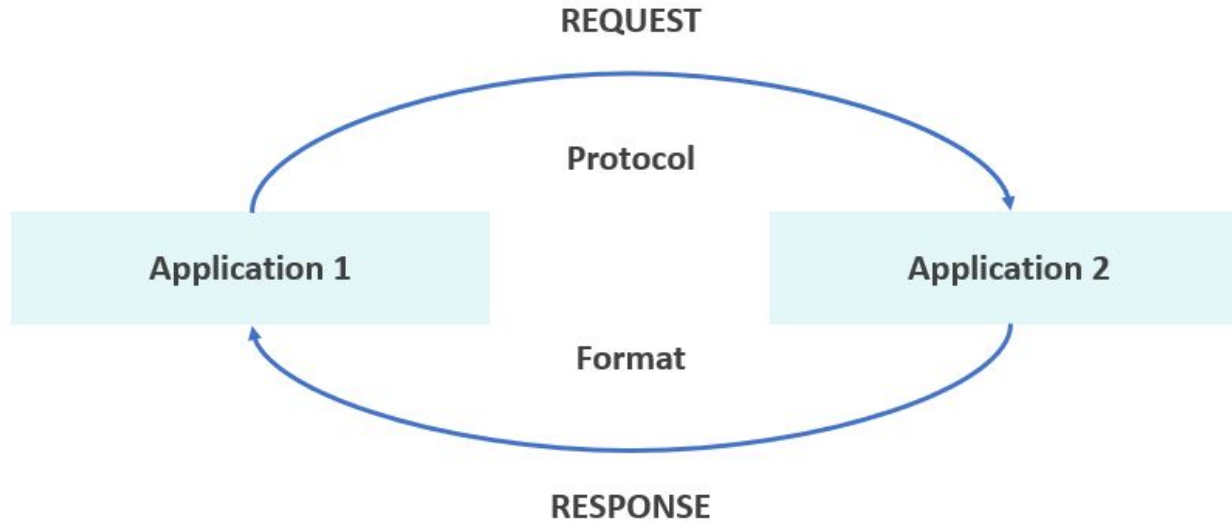


Protocol & Format

The **protocol** is the set of rules that defines how applications can interact with each other, and the **format** specifies how the data can be sent to and accessed by other applications.

Web APIs usually make **HTTP** requests and receive data in the form of a **JSON** or **XML** response.

Rules & Protocols (cont.)



API Architectural Styles



API Architectural Styles

Key aspects:

- **Design Philosophy/Pattern** (e.g. RESTful vs GraphQL)
- **Communication Protocol** (e.g. HTTP vs WebSockets)
- **Encoding** (e.g. Human readable text such as JSON vs Binary formats like Protobuf)

Often, these three different aspects can be mixed together. For example, you can use RESTful API over WebSockets but using a Binary Protocol like Protobuf.



SOAP

SOAP, is the oldest web-focused API protocol that remains in widespread use. Introduced in the late 1990s, SOAP was one of the first protocols designed to allow different applications or services to share resources in a systematic way using network connections.

- XML Based
- Relies on standard protocols, especially HTTP and SMTP, which makes it accessible in any environment, because these protocols are available on all operating systems



REST

REST, short for Representational State Transfer, is an API protocol which was introduced in a 2000 dissertation by Roy Fielding, whose goal was to solve some of the shortcomings of SOAP.

Like SOAP, REST relies on a standard transport protocol, HTTP, to exchange information between different applications or services. However, REST is more flexible in that it supports a variety of data formats, rather than requiring XML. JSON, which is arguably easier to read and write than XML, is the format that many developers use for REST APIs. REST APIs can also offer better performance than SOAP because they can cache information.



JSON RPC and XML RPC

An RPC is a remote procedure call protocol. **XML-RPC** uses XML to encode its calls, while **JSON-RPC** uses JSON for the encoding. Both protocols are simple. A call can contain multiple parameters, and expects one result. They have a couple of key features, which require a different architecture to **REST**:

- They are designed to call methods, whereas REST protocols involve the transfer of documents (resource representations). Or, to put it another way, REST works with resources, whereas RPC is about actions.
 - RPC (`getUserPayments`) vs. REST (`/users/:user_id/payments`)
- The URI identifies the server, but contains no information in its parameters, whereas in REST the URI contains details such as query parameters.
 - RPC (`getUserFailedPayments`) vs. REST (`/users/:user_id/payments?status=failed`)



gRPC

gRPC (gRPC Remote Procedure Calls) is a modern, open source remote procedure call (RPC) framework.

- Created by Google, along with Protocol Buffers.
- HTTP/2 – GRPC is basically a protocol built on top of HTTP/2. HTTP/2 is used as a transport.
- It uses Protocol Buffers as the Interface Definition Language to enable communication between two different systems used for describing the service interface and the structure of payload messages.
- Rest is heavily dependent on HTTP 1.1, and on the other hand, GRPC based on HTTP/2. HTTP 2 is Binary protocol, and HTTP 1.1 is Textual. Binary protocol is much efficient to parse and its safe.



GraphQL

GraphQL is to Facebook what gRPC is to Google: It's an API protocol that was developed internally by Facebook in 2013, then released publicly in 2015. As such, GraphQL, which is officially defined as a query language, also represents an effort to overcome some of the limitations or inefficiencies of REST.

One of the key differences between REST and GraphQL is that **GraphQL lets clients structure data however they want when issuing calls to a server**. This flexibility improves efficiency because it means that **clients can optimize the data they request, rather than having to request and receive data in whichever prepackaged form the server makes available** (which could require receiving more data than the client actually needs, or receiving it in a format that is difficult to use). With GraphQL, the clients can get exactly what they want, without having to worry about transforming the data locally after they receive it.



Apache Thrift


Thrift is a lightweight, language-independent software stack with an associated code generation mechanism for RPC.

Thrift provides clean abstractions for data transport, data serialization, and application level processing.

Thrift was originally developed by Facebook and now it is open sourced as an Apache project. Apache Thrift is a set of code-generation tools that allows developers to build RPC clients and servers by just defining the data types and service interfaces in a simple definition file. Given this file as an input, code is generated to build RPC clients and servers that communicate seamlessly across programming languages.



Comparison Overview

 thriftly.io protocol comparison	First released	Formatting type	Key strength
SOAP	Late 1990s	XML	Widely used and established
REST	2000	JSON, XML, and others	Flexible data formatting
JSON-RPC	mid-2000s	JSON	Simplicity of implementation
gRPC	2015	Protocol buffers by default; can be used with JSON & others also	Ability to define any type of function
GraphQL	2015	JSON	Flexible data structuring
Thrift	2007	JSON or Binary	Adaptable to many use cases

RESTful API



REST

REST is acronym for REpresentational State Transfer. It is architectural style for distributed hypermedia systems and was first presented by Roy Fielding in 2000 in his famous dissertation.

Like any other architectural style, REST also does have it's own 6 guiding constraints which must be satisfied if an interface needs to be referred as RESTful.



REST (cont.)

Guiding Principles of REST:

1. Client-server → By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
2. Stateless → Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
3. Cacheable → Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.



REST (cont.)

Guiding Principles of REST:

4. Uniform interface → Along with the data, server responses should also announce available actions and resources.



REST (cont.)

```
{
  "links": {
    "self": "http://example.com/articles",
    "next": "http://example.com/articles?page[offset]=2",
    "last": "http://example.com/articles?page[offset]=10"
  },
  "data": [{
    "type": "articles",
    "id": "1",
    "attributes": {
      "title": "JSON API paints my bikeshed!"
    },
    "relationships": {
      "author": {
        "links": {
          "self": "http://example.com/articles/1/relationships/author",
          "related": "http://example.com/articles/1/author"
        },
      },
      "comments": {
        "links": {
          "self": "http://example.com/articles/1/relationships/comments",
          "related": "http://example.com/articles/1/comments"
        },
      }
    },
    "links": {
      "self": "http://example.com/articles/1"
    }
  }]
}
```

Just by analysing this single response, a client knows:

1. What was queried (articles)
2. How are structured articles objects (id, title, author, comments)
3. How to retrieve related objects (i.e. the author, list of comments)
4. That there are more articles (10, based on current response length and pagination links)



REST (cont.)

Guiding Principles of REST:

5. Uniform interface → Allows you to use a layered system architecture where you deploy the APIs on server A, and store data on server B and authenticate requests in Server C, for example. A client cannot ordinarily tell whether it is connected directly to the end server or an intermediary along the way.
6. Code on Demand (Optional) → Most of the time, you will be sending the static representations of resources in the form of XML or JSON. But when you need to, you are free to return executable code to support a part of your application, e.g., clients may call your API to get a UI widget rendering code. It is permitted.



RESTful Routing (GET, POST, PUT, DELETE)

- CRUD represents the four basic functions of working with data or resources.
 - (C)reate
 - (R)etrieve
 - (U)pdate
 - (D)elele
- Many applications require some or all users to perform these operations



What is RESTful Routing?

- Uses the concepts of *Resources*
 - Album, Photo
- Allow CRUD operations on the resources through HTTP
- Makes use of the HTTP verbs for these operations
- Makes consistent and “pretty” URL’s



CRUD to REST Mapping

- Create → POST
- Retrieve → GET
- Update → PUT
- Delete → DELETE



RESTful Route Design

HTTP Verb	Path	Controller#Action	Used for
GET	/photos	photos#index	display a list of all photos
GET	/photos/new	photos#new	return an HTML form for creating a new photo
POST	/photos	photos#create	create a new photo
GET	/photos/:id	photos#show	display a specific photo
GET	/photos/:id/edit	photos#edit	return an HTML form for editing a photo
PATCH/PUT	/photos/:id	photos#update	update a specific photo
DELETE	/photos/:id	photos#destroy	delete a specific photo



What is Flask?

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications.

Flask offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use.



What does Flask provide us?

- An HTTP server that listens on a specific port
 - Provides access to the HTTP Request and Response
- A URL Router
 - Maps URL paths to our backend code
- An interface (API) to use and write our own middleware and plugins



Flask

It does not provide:

- A database
- A testing framework
- A file structure

It is lean, mean, and **unopinionated**!



Flask Resources

Flask documentation:

- <https://flask.palletsprojects.com/en/2.2.x/>

Starter tutorials to look at:

- <https://flask.palletsprojects.com/en/2.2.x/quickstart/>

Design Exercise

**Imitation is the sincerest form
of flattery.**

IMDB Clone



The Ask

We are building a simple web application that allows users (movie viewers) to sign up for an account on our web app which allows them to write movie reviews. We want to support the following user actions.

- The user can browse movies and see other reviews for movies.
- The user can post a review for the movies they have seen.
- The user can filter movies by genre, year.

Tasks

- Design the Entities and Relations for this application
- For each entity, only add the attributes necessary to achieve the features described above.

Questions? Concerns?