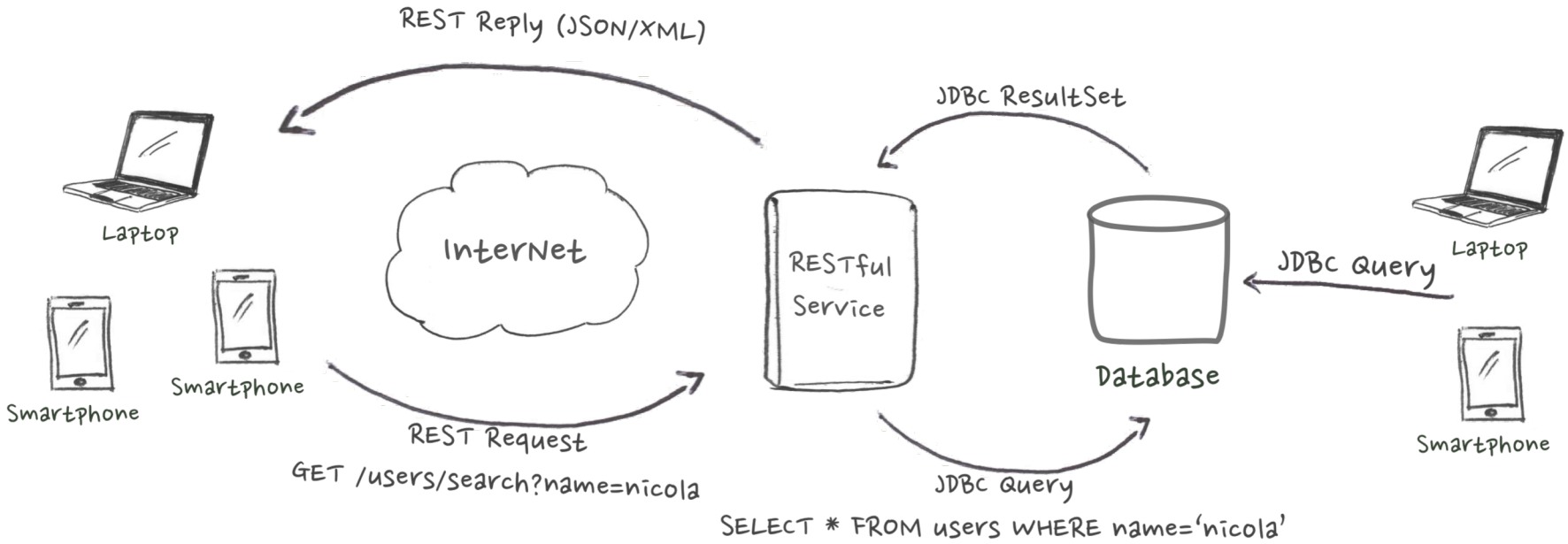
REST (Representational State Transfer)

Università di Modena e Reggio Emilia

Prof. Nicola Bicocchi (nicola.bicocchi@unimore.it)

ING

Why Learn REST?



- REST is used to build scalable Web services (stateless is lightweight)
- REST decouples applications from vendor-specific details (e.g., JDBC requires drivers and knowledge about the underlying database) and prevents exposing DMBS to untrusted networks (e.g. Internet)
- Widely available libraries for many languages (e.g., RESTLet for Java)
- Examples: https://github.com/toddmotto/public-apis

ING

Why Learn REST?

(HTTP Request)

$ curl https://financialmodelingprep.com/api/v3/quote/AAPL

(JSON Reply)

```
[ {
  "symbol" : "AAPL",
  "name" : "Apple Inc.",
  "price" : 276.10000000,
  "changesPercentage" : 2.88000000,
  "change" : 7.73000000,
  "dayLow" : 272.22000000,
  "dayHigh" : 277.85000000,
  "sharesOutstanding" : 4375479808,
  "timestamp" : 1587637985
} ]
```

(HTTP Request)

$ curl https://financialmodelingprep.com/api/v3/company/profile/AAPL

(JSON Reply)

```
{
  "symbol" : "AAPL",
  "profile" : {
    "companyName" : "Apple Inc.",
    "exchange" : "Nasdaq Global Select",
    "industry" : "Computer Hardware",
    "website" : "http://www.apple.com",
    "ceo" : "Timothy D. Cook",
    "sector" : "Technology",
  }
}
```

ING

Why Learn REST?

- A number of mobile apps are built upon RESTful services.
  - https://www.instagram.com/developer/
  - https://developer.twitter.com/en/docs
  - https://developers.facebook.com/docs/graph-api
  - https://www.flickr.com/services/api/
  - https://developer.foursquare.com/

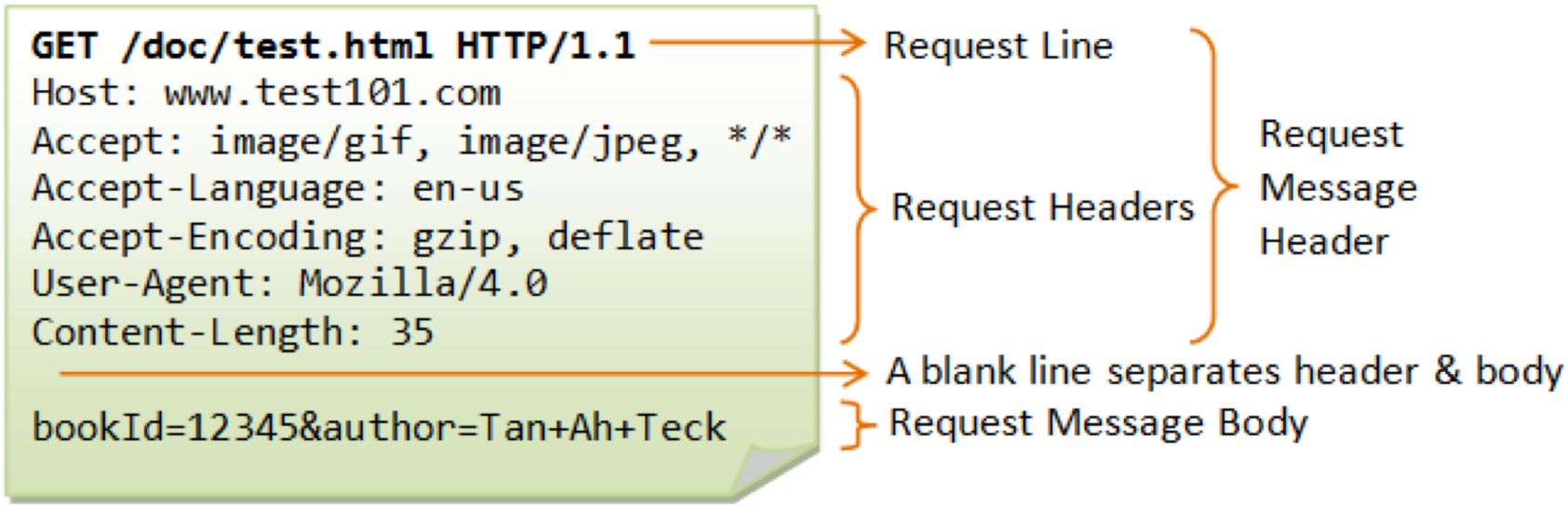
Major Concepts

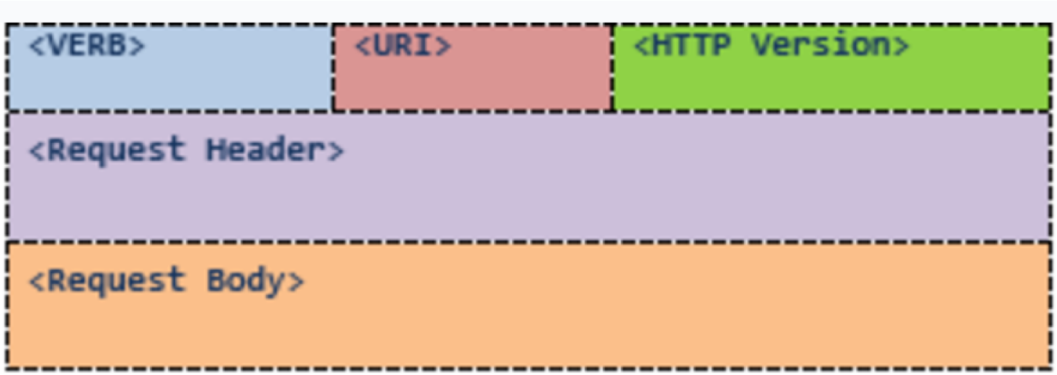
- Messages
- Resources (URIs)
- Representations
- Operations
- *Statelessness*

Messages

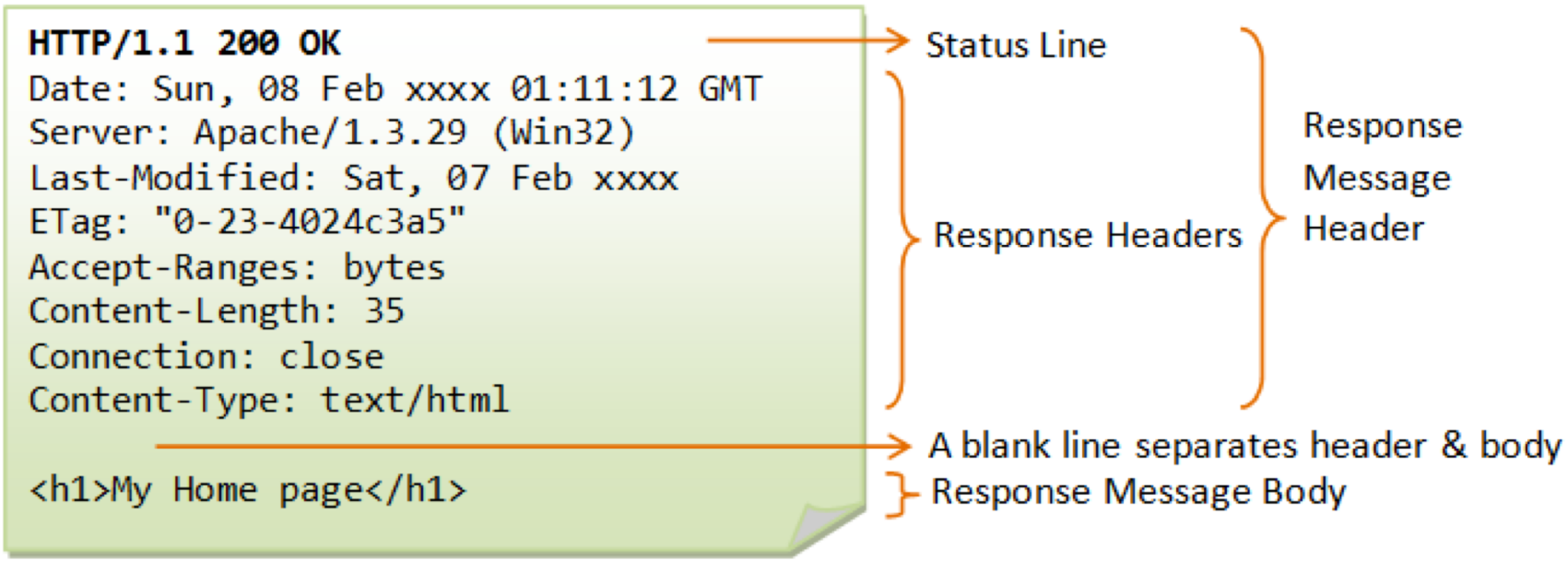
- Clients and REST services talk to each other via messages. Clients send a HTTP request to the server, and the server replies with a HTTP response.
- Request and response contain both metadata and content
- Response content is usually represented in XML or JSON

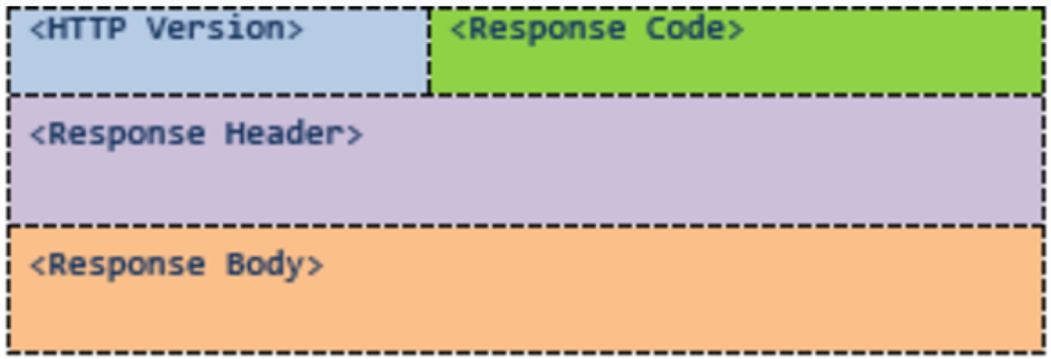
HTTP/1.1 Request





HTTP/1.1 Response





Resources

- Every system uses resources. Resources can be pictures, videos, users data ecc...
- The purpose of a service is to provide access to resources.
- Developers want services to be easy to implement, maintain, extend, and eventually scale up.

Resources

- Resources are identified with specific URLs
- Format: https://servicename/apiversion/resource/id|service
- For example:
  - Place details
    https://api.foursquare.com/v2/venues/VENUE_ID
  - Photos details
    https://api.foursquare.com/v2/photos/PHOTO_ID
  - Search for a user
    https://api.foursquare.com/v2/users/search
  - Recent checkins by friends
    https://api.foursquare.com/v2/checkins/recent

Representations

- The focus of a RESTful service is on resources and how to provide access to these resources. A resource can be thought of as an object as in OOP. A resource can consist of other resources.
- While designing a system, the first thing to do is identify the resources and determine how they are related to each other.
- This is similar to designing a database or object oriented software: identify key entities and their mutual relations.

Representations

- Once resources have been identified, it is important to properly represent resources (for example, the toString() method represent resources using a plain String).
- You can use any format for representing the resources as REST does not put any restrictions.
- Nevertheless, the most used representations are XML and JSON

Representations

Listing One: JSON representation of a resource.

```
{
    "ID": "1",
    "Name": "M Vaqqas",
    "Email": "m.vaqqas@gmail.com",
    "Country": "India"
}
```

Listing Two: XML representation of a resource.

```
<Person>

<ID>1</ID>

<Name>M Vaqqas</Name>

<Email>m.vaqqas@gmail.com</Email>

<Country>India</Country>
</Person>
```

Operations (HTTP Verbs)

- HTTP Verbs (see HTTP Request) define operations on specific resources.

- GET /users/145 *(retrieve user 145)*
- DELETE /users/145 *(delete user 145)*
- POST /users/ *(add a new user)*
- PUT /users/17 *(update user 17)*

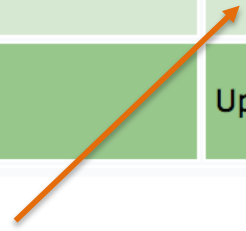
Operations (HTTP Verbs)

- GET Read a resource
  - Safe
- PUT Insert/update a resource
  - Idempotent
- POST Insert/update a resource
  - N/A
- DELETE Delete a resource
  - Idempotent

- A Safe HTTP method does not make any changes to the resource on the server.
- An Idempotent HTTP method has same effect no matter how many times it is performed.
- Classifying methods as Safe and Idempotent makes it easy to predict the results in unreliable environments such as the Web (clients may fire the same request multiple times for example)

PUT and POST

Request	Operation
`PUT http://MyService/Persons/`	Won't work. `PUT` requires a complete URI
`PUT http://MyService/Persons/1`	Insert a new person with `PersonID=1` if it does not already exist, or else update the existing resource
`POST http://MyService/Persons/`	Insert a new person every time this request is made and generate a new `PersonID`.
`POST http://MyService/Persons/1`	Update the existing person where `PersonID=1`



Idempotent fails here!

Addressing resources (URIs)

- REST requires each resource to have at least one URI
- RESTful services uses a directory hierarchy to address resources
- The job of a URI is to identify a resource or a collection of resources
- The actual operation is determined by an HTTP verb. The URI should not say anything about the operation or action
- **Protocol://ServiceName/ResourceType/ResourceID**

Addressing resources (URIs)

- Use plural nouns for naming your resources.
- Avoid using spaces as they create confusion. Use an _ (underscore) or – (hyphen) instead.
- A URI is case insensitive. I use camel case in my URIs for better clarity. You can use all lower-case URIs.
- A cool URI never changes; so give some thought before deciding on the URIs for your service. If you need to change the location of a resource, do not discard the old URI and redirect the client to the new location.
- Avoid verbs for your resource names. Verbs are more suitable for the names of operations.

Query parameters

- The basic purpose of query parameters is to provide parameters to an operation that needs the data items.
  - http://MyService/Persons/1?format=json
  - http://MyService/Persons/search?name='nicola'

- Avoid this!
  - http://MyService/Persons/1/json/

Statelessness

- A RESTful service is stateless and does not maintain the application state for any client.
- A request cannot be dependent on a past request. A REST service treats each request independently.

Statelessness

Stateless design

Request1: GET http://MyService/Persons/1 HTTP/1.1

Request2: GET http://MyService/Persons/2 HTTP/1.1

Stateful design (Dangerous! Which client??)

Request1: GET http://MyService/Persons/1 HTTP/1.1

Request2: GET http://MyService/NextPerson HTTP/1.1

Documentation

- There is no excuse for not documenting your service.
- You should document every resource and URI for client developers. You can use any format for structuring your document, but it should contain enough information about resources, URIs, Available Methods, and any other information required for accessing your service.

Documentation

Service Name: MyService

Address: http://MyService/

Resource	Methods	URI	Description
Person	GET,POST,PUT, DELETE	http://MyService/Persons/{PersonID}	Contains information about a person {PersonID} is optional **Format:** text/xml
Club	GET,POST,PUT	http://MyService/Clubs/{ClubID}	Contains information about a club. A club can be joined my multiple people {ClubID} is optional **Format:** text/xml
Search	GET	http://MyService/Search?	Search a person or a club **Format:** text/xml **Query Parameters:** Name: String, Name of a person or a club Country: String, optional, Name of the country of a person or a club Type: String, optional, Person or Club. If not provided then search will result in both Person and Cubs

Criticism

- No transactions support
  - DBMS (usually behind REST services) support transactions
- No publish/subscribe support.
  - Notification is done by polling.
  - The client can poll the server. GET is extremely optimized on the web.
- High bandwidth
  - HTTP uses a request/response model, so there's a lot of baggage flying around the network to make it all work.

Advantages

- REST is a great way of developing lightweight Web services that are easy to implement, maintain, and discover.
- HTTP provides an excellent interface to implement RESTful services with features like a uniform interface and caching. However, it is up to developers to implement and utilize these features correctly.
- If we get the basics right, a RESTful service can be easily implemented using any of the existing technologies such as Python, .NET, or Java.