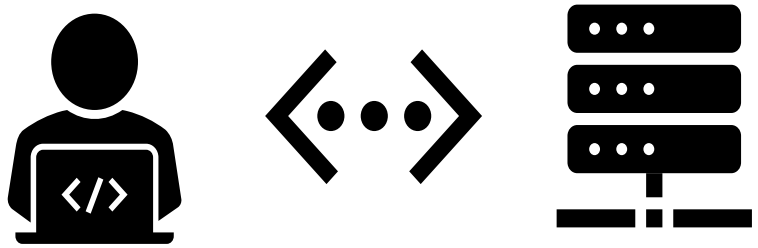
REST Basics



{ JSON }

http://

SWEN-261
Introduction to Software Engineering

Department of Software Engineering
Rochester Institute of Technology

Software Engineering
Rochester Institute of Technology

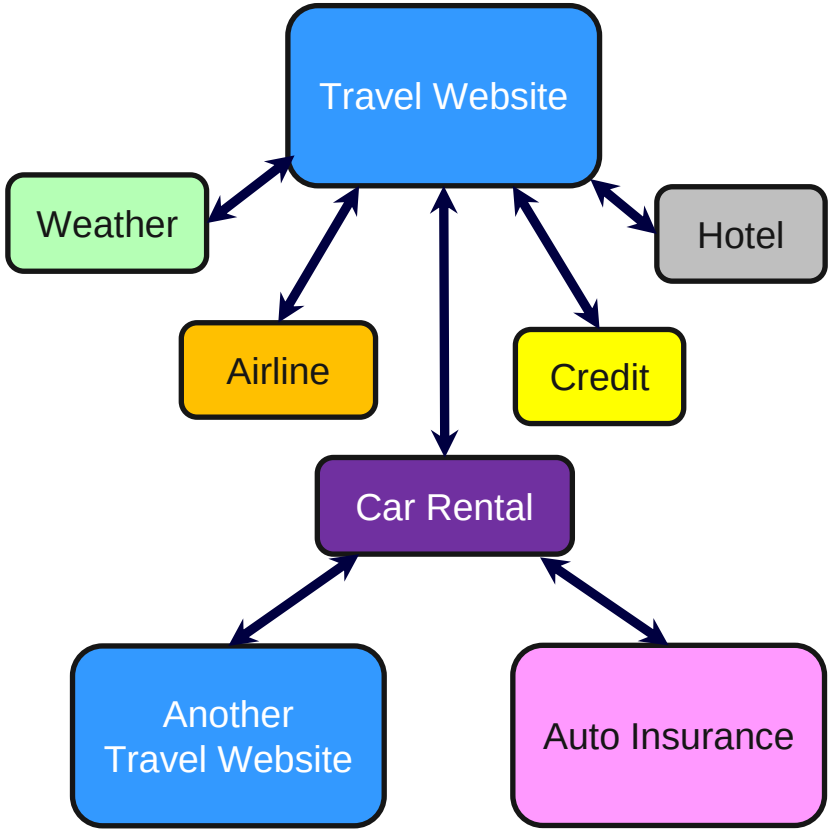
What is an API?

- APIs (Application Program Interfaces) allow applications to communicate with one another
  - ***Applications that communicate via APIs can be located on the same computer, over a local network, or over the internet***
- An API is a contract between a client application and a service application
  - ***The client application sends a request in an agreed upon format to the API of the service application***
  - ***The service application API sends a response back to the client in an agreed upon format***
  - ***Neither the client application nor the service application need to know the implementation details of the other***
- APIs allow access to resources while maintaining security and control

APIs in Action

Consider a travel website

- ***Its "product" is a one-stop shop for a customer's travel needs***
- ***The travel company itself does not directly provide nor control the travel services***
- ***It must rely on other companies for these services, needs access to their data, schedules, etc, and does so via APIs***
  - Weather – warn customers of advisories and warnings
  - Airline – compare fares and schedules, book flights
  - Car Rental – compare rates and availability, reserve cars
  - Hotel – compare rates and availability, reserve rooms
  - Credit – payments



Now consider the benefits the service provider gains by making an API available

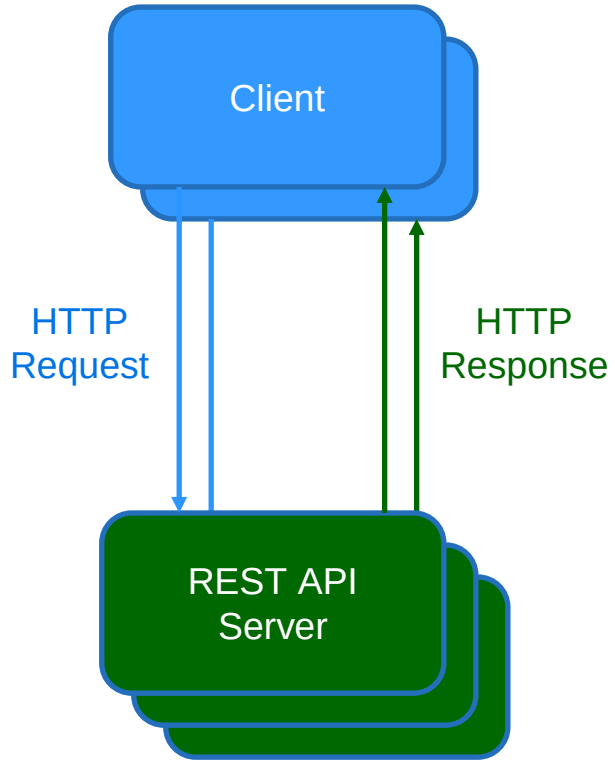
- ***By integrating its services into the travel website, it increases the opportunity for sales***
- ***With little or no extra development, it can expand its presence***
  - Other travel websites
  - Other industries
    - For example, a Car Rental company can make its services and data available to an auto insurance company for clients whose car has been in an accident

What is REST?

- REpresentational State Transfer – an architectural standard for accessing and modifying resources
- A REST server provides access to resources via standard HyperText Transfer Protocol (HTTP) methods
- A REST API is stateless which means it is a client's responsibility to maintain state and pass this state with each request
- A resource is identified by a Uniform Resource Identifier (URI), which looks very similar to a website URL
- REST APIs define a set of functions in which the developers can perform requests and receive responses
- First introduced by Roy Fielding in his 2000 doctoral dissertation entitled "Architectural Styles and the Design of Network-based Software Architectures"

Why REST?

- Maintains separation between client and server
  - ***The same interface can be used whether the client is a user interface or another REST API server***
- Provides a uniform interface to access and manage resources
- Scalability and Reliability
  - ***REST APIs can be deployed to multiple servers in different locations***
  - ***If one server becomes unavailable, requests can be automatically routed to another with no loss in service (load balancing)***
  - ***As request volumes increase, additional REST API servers can be brought online***
- Language and Platform Independence
  - ***REST APIs can be written in nearly any language and clients can be written in a completely different language***
  - ***REST APIs can be hosted on nearly any Operating System***
- Flexible Data Formats
  - ***REST APIs can accept and return multiple data formats, e.g. JSON, XML***



Resources and URIs

- A resource is identified by a Uniform Resource Identifier
- A URI looks very similar to a website address
- The basic format is

scheme :// host:port / path to resource ? query parameter

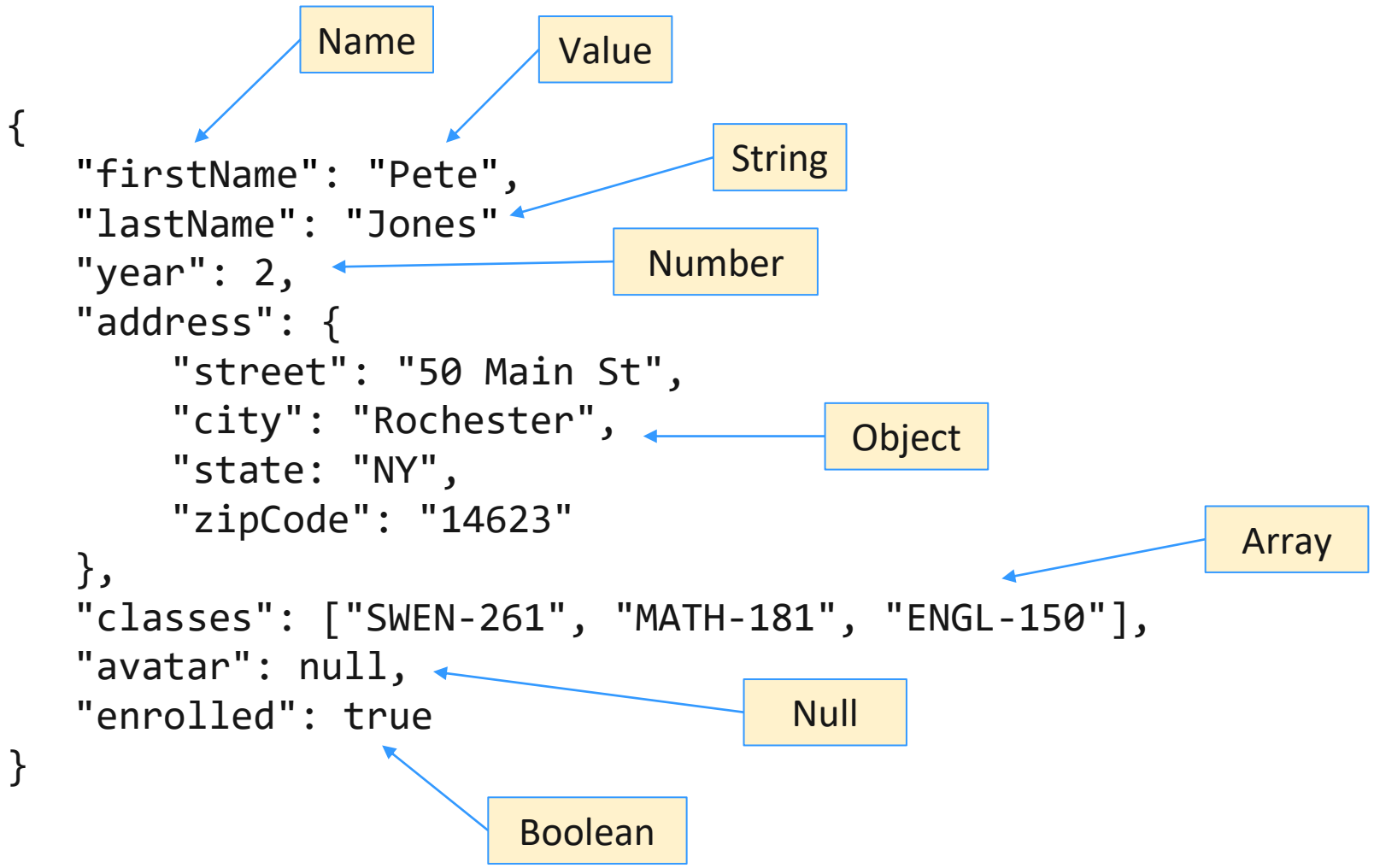
- Example

http :// www.state.edu:9150 / se/faculty ? id=310

JSON

- JavaScript Object Notation – a human readable data interchange format for defining and transmitting objects
- The syntax supports name-value pairs and arrays
  - ***A name is surrounded by double quotes and separated from the value by a colon***
  - ***A value can a number, string, boolean, array, object, or null***
  - ***An array is surrounded by square brackets***
- Curly braces wrap objects
- Commas separate name-value pairs and array elements
- Names follow the camel case convention

JSON Example: Student



REST HTTP Methods

- The most commonly used HTTP methods in REST carry out CRUD operations (Create, Read, Update, Delete)
  - ***POST – Create a new resource***
    - `POST /petstore/pets/dog` Create a new dog
  - ***GET – Read access to a resource***
    - `GET /petstore/pets` Get all pets
    - `GET /petstore/pets/dog/{id}` Get a specific dog
  - ***PUT – Update or create a resource***
    - `PUT /petstore/pets/dog/{id}` Update a specific dog
  - ***DELETE – Delete a resource***
    - `DELETE /petstore/pets/dog/{id}` Delete a specific dog (because it went to a loving home)
- When you type a URL into a browser, an HTTP GET request is sent to the website and the response data is used to render the page

REST Request Components

- A REST API request consists of four main parts:
  - ***Method***
    - Required
    - Identifies the operation
    - Example: `GET`
  - ***URI***
    - Required
    - Identifies the resource
    - May include query parameters to identify specific content or actions
    - Example: `http://localhost:8080/jedi/5`
  - ***Headers***
    - Optional, but generally used
    - Provides additional information about the request or client
    - Applications, e.g. Browsers, or frameworks, e.g. Spring, often add standard and their own headers to requests
      - For the purposes of the term project, we will focus on `Content-Type` and custom headers
    - Example: `Content-Type: application/json`
  - ***Body***
    - Generally used for POST and PUT, but not for GET and DELETE
    - Representation of one or more objects
    - Example: `{"id": 3, "lastName": "Skywalker", "firstName": "Anakin"}`

REST Response Components

- A REST API response consists of three main parts:
  - ***Status Code***
    - Required
    - Identifies the result of the operation
    - Example: 200/OK
  - ***Headers***
    - Optional
    - Provides additional information about the response to the client
    - Example: `api-status-code: 3`
  - ***Body***
    - Required for GET, but often used for other methods
    - Representation of one or more objects
    - Example: `{"id": 3, "lastName": "Skywalker", "firstName": "Anakin"}`
- Common HTTP Status Codes
  - **`200/OK`** – ***Request was completed successfully***
  - **`201/Created`** – ***Resource was created successfully***
  - **`400/Bad Request`** – ***Body of request was invalid***
  - **`403/Forbidden`** – ***Caller does not have permissions for the requested resource***
  - **`404/Not Found`** – ***Requested resource could not be found***
  - **`500/Internal Server Error`** – ***Server cannot fulfill request and does not want to expose specifics to client***
  - **`501/Not Implemented`** – ***Requested method is not currently supported***

REST HTTP Methods - POST

- Creates a new resource
- Request
  - ***URI specifies the resource to be created***

    `http://localhost:8080/jedi`
  - ***Header tells the REST API the format of the Body***

    `Content-Type: application/json`
  - ***Body is a representation of the jedi object***

```
{
   "lastName": "Skywalker",
   "firstName": "Anakin“
}
```

    Notice the `"id"` field is not included - The unique identifier of a resource should be created and managed by the REST API service unless a field is determined to be unique
- Response
  - ***Common Status Codes***

    `201 – CREATED`

    `403 – FORBIDDEN`
  - ***Header***

    Application dependent
  - ***Body is a representation of the created object***

```
{
   “id": 3,
   "lastName": "Skywalker",
   "firstName": "Anakin“
}
```

REST HTTP Methods - GET

- Retrieves a resource
- Request
  - ***URI provides enough information identify the resource***

    ```
    http://localhost:8080/jedi/3
    ```
  - ***Header***

    Generally not applicable
  - ***Body***

    Generally not applicable
- Response
  - ***Common Status Codes***

    ```
    200 – OK
    404 – NOT FOUND
    ```
  - ***Header***

    Application dependent
  - ***Body is a representation of the object requested***

    ```
    {
      "id": 3,
      "lastName": "Skywalker",
      "firstName": "Anakin"
    }
    ```

    If multiple objects are requested, an array would be returned

REST HTTP Methods - PUT

- Update a resource or create the resource if it does not exist
- Request
  - ***URI provides enough information identify the resource***

    ```
    http://localhost:8080/jedi
    ```
  - ***Header tells the REST API the format of the Body***

    ```
    Content-Type: application/json
    ```
  - ***Body of the request contains an object with the fields to be updated***

    ```
    {
      "id": 3,
      "lastName": "Vader",
      "firstName": "Darth"
    }
    ```
- Response
  - ***Common Status Codes***

    ```
    200 – OK
    404 – NOT FOUND
    ```
  - ***Header***

    Application dependent
  - ***Body is a representation of the updated object***

    ```
    {
      "id": 3,
      "lastName": “Vader",
      "firstName": “Darth"
    }
    ```

REST HTTP Methods - DELETE

- Deletes a resource
- Request
  - ***The URI specifies the resource to be deleted***

    `http://localhost:8080/jedi/3`
  - ***Header***

    Generally not applicable
  - ***Body***

    Not Applicable
- Response
  - ***Common Status Codes***
    - `200 – OK`
    - `404 – NOT FOUND`
  - ***Header***

    Application dependent
  - ***Body***

    Not applicable

Accessing a REST API

- Write a client application
- Use an existing tool
  - ***Two of the most popular tools are***
    - cURL (client URL) – a command-line tool available by default in most operating systems including Windows, Mac, and Linux
    - Postman – a graphical user interface for API testing (www.postman.com)

Serialization and Deserialization

- As we've seen, JSON is a human-readable text format
- In our REST API application, we do not want to deal with text, but rather Java objects
  - ***From the previous HTTP examples, you can envision Jedi being a class with 3 fields:***
    - `id – Number`
    - `firstname – String`
    - `lastname – String`
- Serialization is the process of converting an application object (e.g. Java object) to text (or byte stream)
- Deserialization is the reverse – converting text (or byte stream) into an application object
- HTTP POST and PUT *requests* contain a JSON Object (text representation) that needs to be converted to an application object our REST API application code can work with
- Conversely, GET *responses* from our REST API need to be converted from an application object into a JSON object that can be transmitted back to the client
- Additionally, within a REST API service, we usually want to persist data, whether it be in a file, database, or other storage
  - ***As information is typically represented in files and database as text, serialization and deserialization can be used to transform application objects to JSON objects and vice-versa***
  - ***The JSON objects, which are text, are then easily written to and read from a file, a database, etc***

REST API Frameworks

- Nearly every language has REST frameworks available, most are open source, that support rapid and reliable development
- We will use Java and the Spring Boot framework in our term project
  - ***Spring Boot provides the scaffolding for stand-alone, light-weight, production-grade REST API applications***
  - ***Includes an embedded Tomcat server that hosts your APIs and makes them available to clients on a network***
  - ***Routes HTTP requests to your class methods for handling***
  - ***Built-in support for serialization and deserialization***
  - ***The Spring Initializr wizard, available at start.spring.io or via VSCode extension, quickly builds a baseline project***
    - You will not need to use Spring Initializr as the starter projects are provided
  - ***Many annotations, e.g. `@RestController`, are available to easily control configuration***
  - ***See the course resources page for more information and helpful links***