Working with REST APIs in SENG Workshop 3

Fethi Rabhi

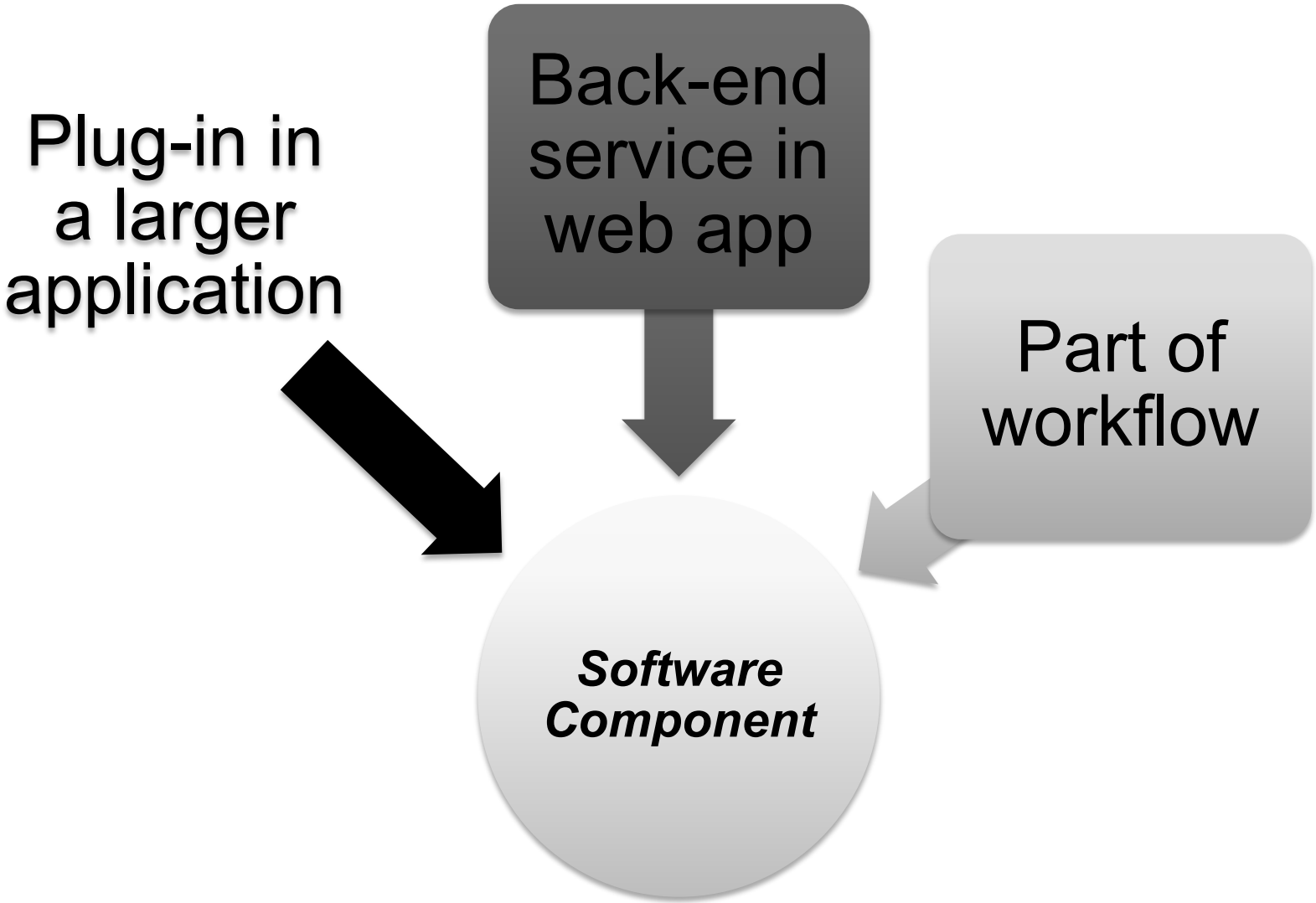
Acknowledgement: to all SENG3011 mentors who have helped putting these slides together

Software components

Designing Software Using Components

- High quality code
  - Modular
  - High cohesion
  - Low coupling
- Many technologies available for developing components
  - Library components (C# DLL file, JAR File etc.)
  - Components are an essential part of web services

Multiple reuse of a component



Example 1: Java component technology

- Components packaged as .jar files
- To create a .jar file
  - Export from IDE (e.g. Eclipse)
  - Use command line:
    - jar cf jar-file input-file(s)
- Available with popular build tools:
  - Maven
  - Ant
  - Buildr
  - … …

Example 2: C# and .NET

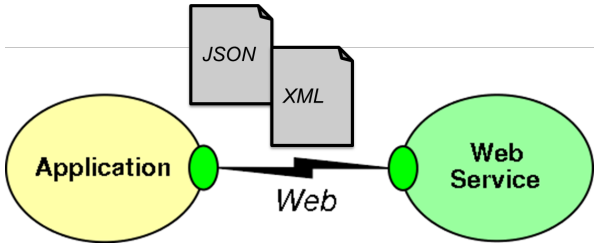
- Component technology for Windows systems
- DLL = basic component that can be executed by a Windows application
- Many utilities for creating and managing components
- DLL lifecycle
  - Create C# Classes
  - Generate DLL file
  - Generate EXE file
  - Run the EXE file

Service Oriented Computing

The Concept of Programmable Web

- The Programmable Web use the same technologies and communication protocols of the WWW
- Difference:
  - The data is not delivered necessarily for human consumption
  - A client can be implemented using any programming language
- Technologies
  - Services and APIs
  - Transport protocol: Hyper Text Transfer Protocol (HTTP)
  - Clients: Browser, Java, Web API, …
  - Data serialization languages

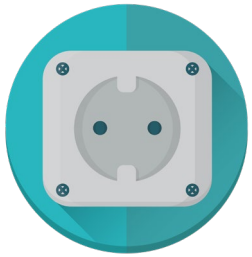
Web Services



- ‘logical units with clearly defined interfaces(API):’
  - What functionality they perform
  - Which data formats they accept and produce
- They are application independent
- Services can be used by other services and applications
- Web services are not prepared to human consumption (in contrast to websites).
  - Web services require an architectural style to provide clear and unambiguous interaction (clearly defined interfaces).

Web API

- Application Programming Interfaces
  - A good analogy is the electricity wall socket
- Endpoints addressable over the Web are called Web APIs.
- How the service is exposed:
  - Protocol semantics
  - Application semantics
- We frequently use Web API instead of Web services but they are not the same
- There are many technologies that support Web Services (we will be focusing on the RESTfull Web API)

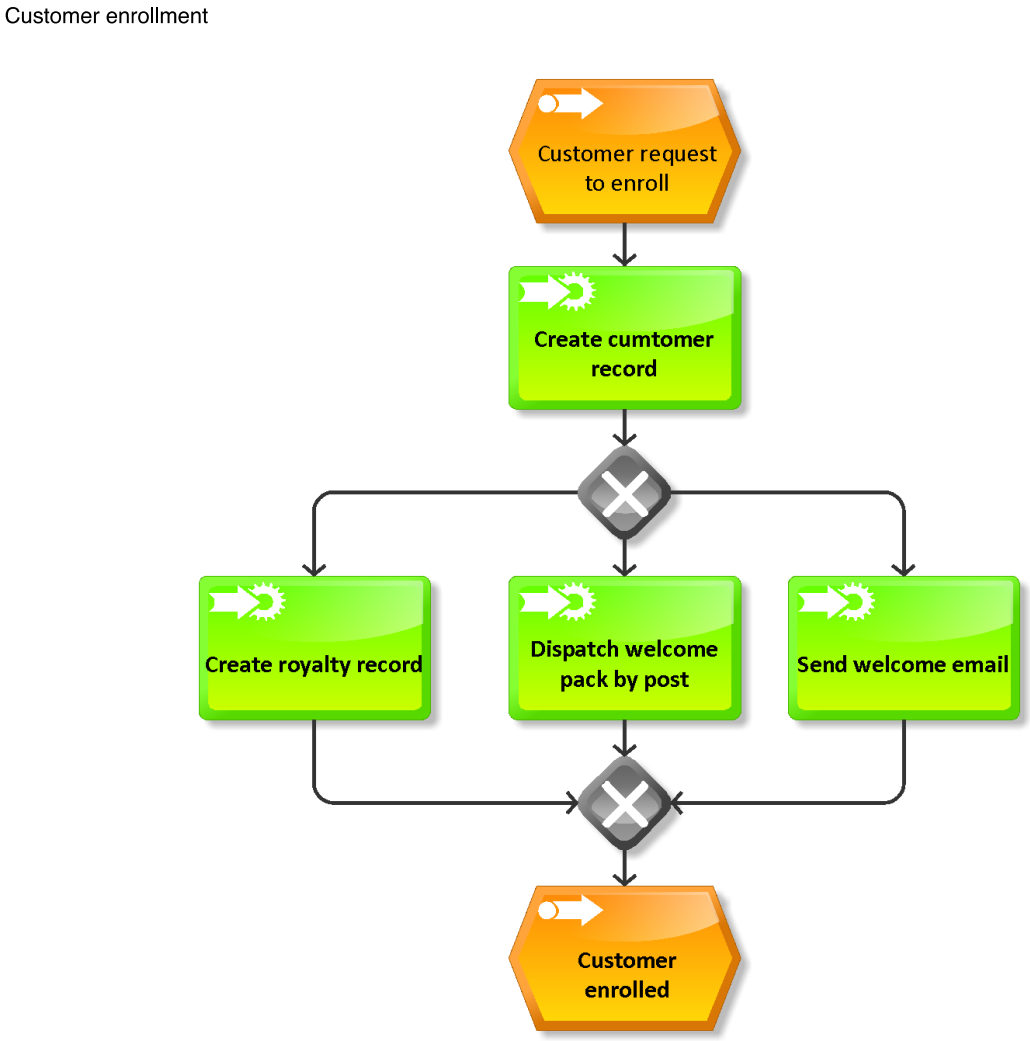


- Service: Electricity
- Conforms to specs: 220V, 60Hz …
- Fitting patterns are defined
- Through the standard interface all connecting equipment (consumers) work
- A layer of abstraction

Market Impact

- Making functionality available over the web changed the way software functionality delivered.
- If you needed a CRM functionality in 1990s you had to invest in hardware, software, the CRM experts, training …
- Today's CRM providers like Salesforce use cloud to deliver the functionality.
  - Multi-tennacy – sharing common infrastructure among customers.
  - Using web browsers was the norm to access this functionality
  - Today customers are granted API level access
    - Non salesforce applications can easily use the services.
- Thousands of companies are changing their strategies toward delivering functionality through Web APIs:
  - https://www.programmableweb.com/apis/directory is a good source

Example of a process using multiple web services



Designing complex systems using Web services

- Using Web services must be done with care
- Issues when defining individual services
  - Synchronous vs asynchronous services
  - State management and scalability
  - Data management
- Linking multiple services together
  - Can use Business Process Management Framework
    - BPEL/BPMN
  - Can use a workflow language
    - TAVERNA
- Several design patterns exist

Synchronous Web services

- More suitable:
  - where real-time interaction with minimal delays is needed,
  - where subsequent actions are dependent on the response received for the previous message transferred,
  - further actions need to be performed in sequential manner.
- Example:
  - ATM machine need to interact with the back-end system to check the available balance.

Asynchronous web services

- More suitable:
  - where systems have long running jobs and there is no need of real-time responses.
  - when you need low latency – blocking a call may slow the system
- Example:
  - An ERP system needs to publish some information so that any interested parties can subscribe to that and get the updates.

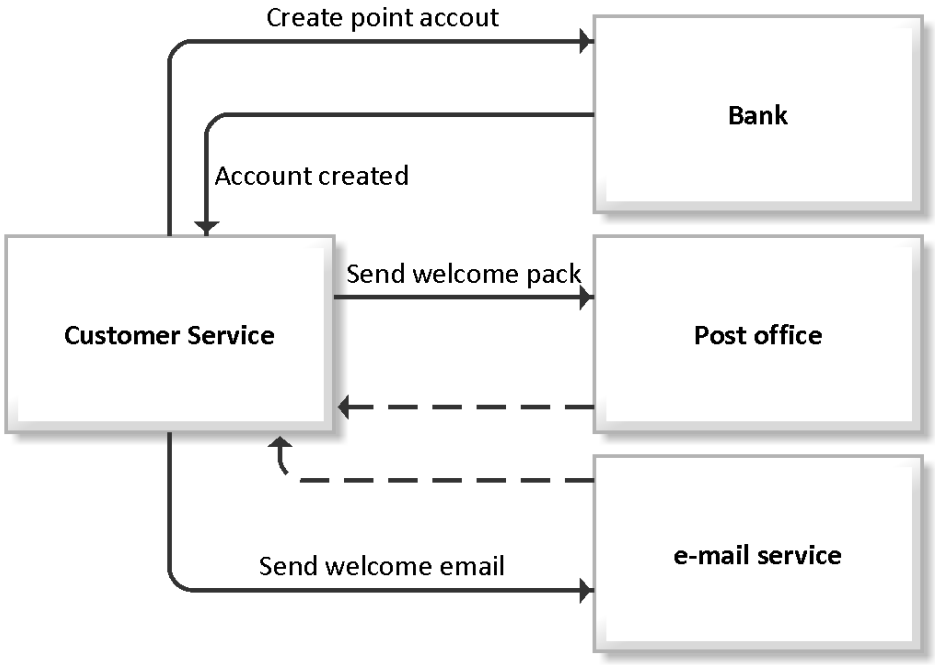
Separation of Stateful from Stateless

- Stateless:
  - Deals with behavior, pure business logic
  - Sending an email
  - Displaying the fuel consumption for the moment
  - HTTP protocol
- Stateful:
  - Deals with keeping records of things
  - Expecting an acknowledgement for the email sent
  - Displaying the average fuel consumption for a period.
  - FTP protocol

Scaling up

- Decoupling behaviour from state enable us to scale up the stateless processes.
- Scaling up stateless processes is easy.
  - You can run KM to Miles Conversion on multiple nodes easily
  - Various platforms exists: AWS Lambda is a popular example
- Scaling up the stateful part is difficult
  - The aggregate is the only strongly consistent truth
  - Single active instance can run at a time
  - Usually scaled up by using active/passive availability clusters
    - (Establishing fully redundant instances of nodes, brought online when its associated primary node fails)

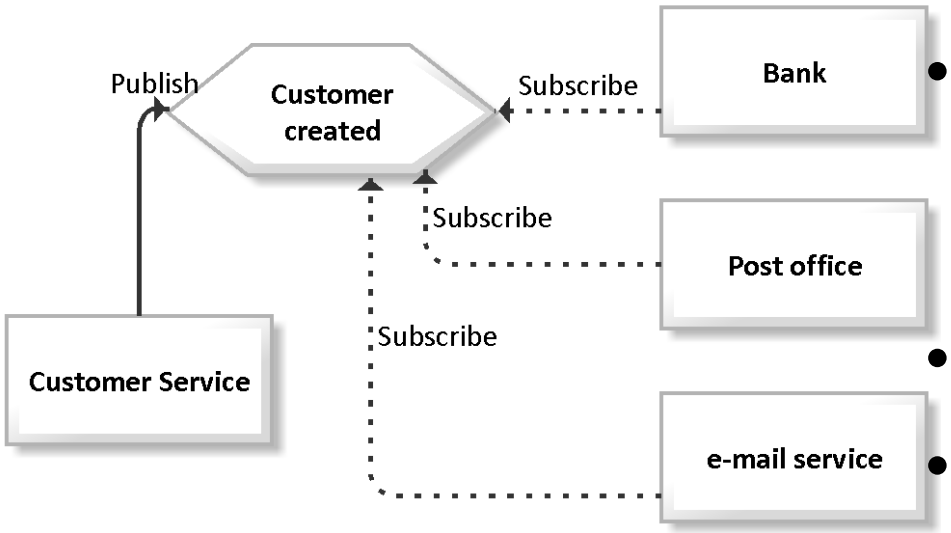
Orchestration



- Create a central control mechanism within:

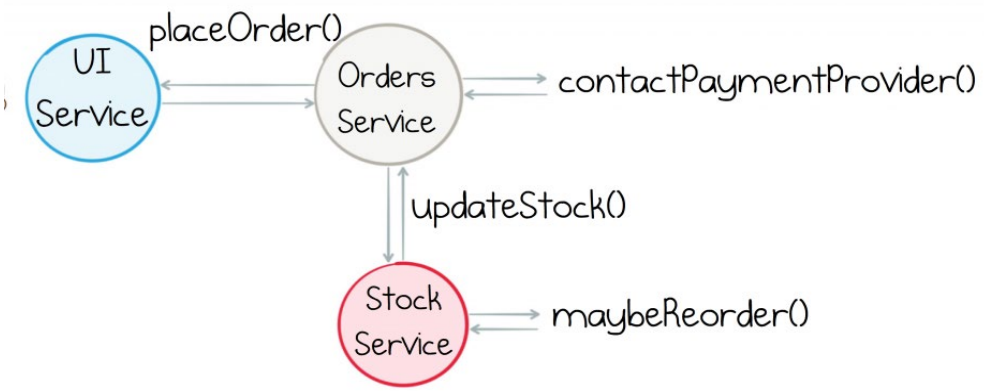
  CustomerService
- Once the process initiated CustomerService send request to other services.
- We can model into code or use BPM software.
- - Tightly coupled
- - High cost to change
- + Can monitor the status of the process.

Choreography



- Customer Service created the event.
- All services subscribe to this event react to it.
- + Loosely coupled
- + Easy to change
- - Additional work is needed to monitor the status of the process.

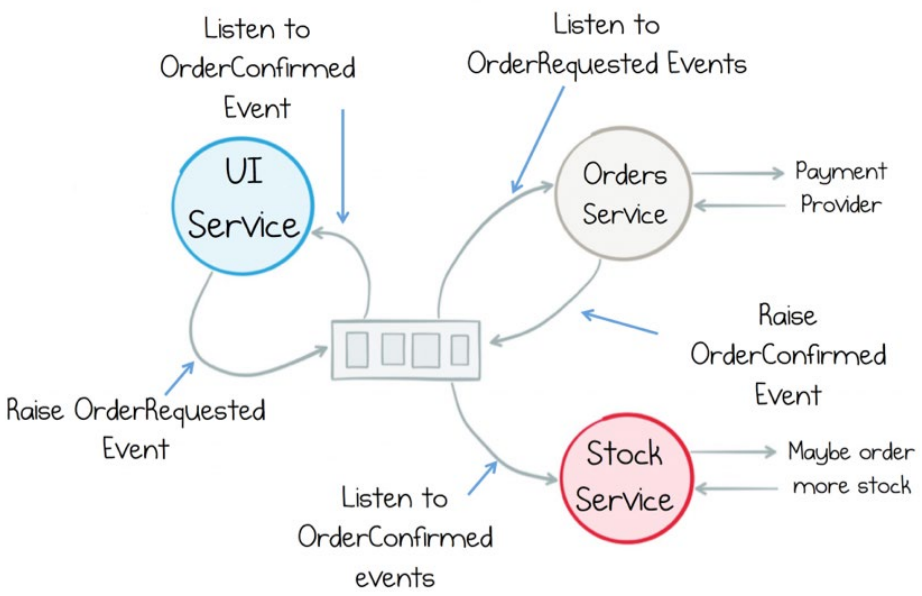
Request/Response Collaboration Pattern



1-Customer orders an item
2-Payment is processed
3-The system check the availability and the need for reorder

- Well aligned with synchronous communication
- For asynchronous applications adaptation is required:
  - Start the operation
  - Register a call back
    - ask server to notify when the operation complete

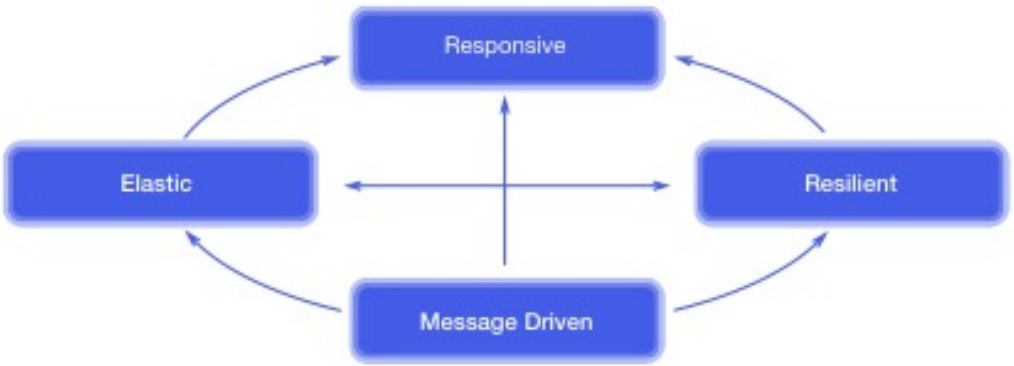
Event based collaboration pattern



- The UI Service raises Order-Requested event
- Orders Service and the Stock Service react to the raised event.
- Order service raise Order-Confirmed event
- UI Service reacts to Order-Confirmed

- Process announce what happened
- Other services decides what to do
- Business logic is distributed
- Highly decoupled – can add new services easily.

Reactive Systems



- Systems that are* :
  - Responsive,
  - Resilient,
  - Elastic and
  - **Message Driven**
    - Asynchronous, nonblocking message-passing that establish a boundary between components…
    - that ensures loose coupling, isolation and location transparency.

Microservices

- Microservices are services modeled after a business domain
- Conwey's Principle:
  - Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization's communication structure
- Information Systems Department of an Army:
  - How will the communication structure shape?
    - Command and control
  - Who will be the project manager?
    - The highest ranking officer
- A startup ? Will you give the same answers?

REST APIs

Representational State Transfer (REST)

- A way of providing interoperability between computer systems on the Internet.
  - REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations.
- An architectural style of building networked systems
  - a “design guideline” for building a system (or a service in our context) on the Web
  - defines a set of architectural constraints in a protocol
- REST is built on standards:
  - HTTP, URL, XML/HTML/JPEG/ … (resource representations)
  - text/xml, text/html, image/gif, image/jpeg, … (MIME Types)
- REST itself is not an official standard specification

REST video: https://www.youtube.com/watch?v=7YcW25PHnAA)

Resource Oriented Architectures

- ROA:
  - Architecture for creating Web APIs that conforms to the REST design principles
  - Base technologies: URLs, HTTP and Hypermedia
- Web Services with a ROA architecture are called RESTful Web Services (Restful Web APIs)
- HTTP requests are used to manipulate the state of a *resource*
  - A resource is something that can be stored on a computer and represented as a stream of bits.
  - Resource-Oriented refers to modelling each entities as Resources which can be accessed by at least one identifier.

URI: Identifies the resource (SENG3011) to manipulate

http://www.unsw.edu.au/course/SENG3011

HTTP method: The action to be performed to manipulate the resource

Resource definition

- A thing that users want to create a link to, retrieve, annotate, or perform other operations on.
- A resource:
  - is **unique** (i.e., can be identied uniquely)
  - has at least one **representation**,
  - has one or more **attributes** beyond ID
  - has a potential **schema**, or definition
  - can provide **context**
  - is reachable within the **addressable** universe
- collections, relationships (structural, semantic)

ROA Properties

- Addressability - Every object and resource in your system is reachable through a unique identifier
- Uniform interface – Deals with how a client talk to a service and understands what to tell the service. Also the service should be able to understand what clients wants to say.
- Statelessness - All calls from clients are independent, every HTTP request happens in a complete isolation.
- Connectedness - APIs become more self-descriptive and discoverable when links are returned in the response

Most common REST operators

GET

Retrieve a representational of resource (without changing it)

PUT

Create or replace a resource by supplying representational to it

DELETE

Ensure that a given resource is no longer exist

POST

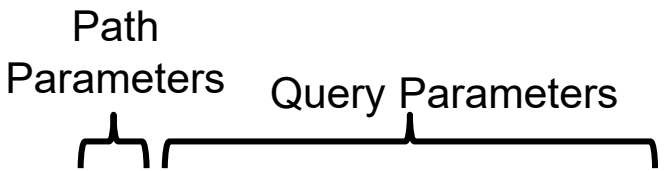
Augment a resource with additional representational

REST Parameters

- REST APIs have four types of parameters:
  - Header parameters: Parameters included in the request header, usually related to authorization.
  - Path parameters: Parameters within the path of the endpoint, before the query string (?). These are usually set off within curly braces.
  - Query string parameters: Parameters in the query string of the endpoint, after the ?.
  - Request body parameters: Parameters included in the request body. Usually submitted as JSON.

REST Example

- HTTP Method
  - POST
- Request URL
  - http://<myserver.ulr>/unsw/examples/001?course=seng3011&term=20T1



- Request Header
  - Content-Type: application/json
  - Accept: text/html

Content type that is used in the body of request.

Content types that are valid in the response message.

- Request body

```
{
"course_code": "seng3011",
”title": "Software Engineering Workshop 3",
"url": ”http://cse.unsw.edu/au/~se3011"
}
```

- Response

Status code

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
.
.
</html>
```

Resource Representation

- **A resource needs a representation for it to be sent to the client**

a representation of a resource - some data about the ’current state’ of a resource

E.g., On a library system, books can have representations in :–

- XML files
- web pages
- Json files
- printer-friendly-format, etc.

when a representation of a resource may also contain metadata about the resource (e.g., books: book itself + metadata such as cover-image, reviews, other related books) - relationships.

Representations can flow the other way too: a client send a new or updated ’representation’ of a resource and the server creates/updates the resource.

Response Codes

- Using proper status codes. Using them consistently in your responses will help the client understand the interactions better.
- The HTTP specification has a guideline for the codes
- Utilize these codes but restrict the number of codes used for clean/clear responses.
- Few examples:

Code	Description	When
200	OK	All good
304	Not modified	cached
404, 401, 403	Not found, Unauthorized, Forbidden	For authentication and authorization

Response Format

- Response format of API is designed for client's needs
- Should support multiple formats and allow the client content negotiation (i.e. Content-Type)
- Use simple objects.
- Request for a single resource should return a single object.
- Request for multiple resources can return a collection - wrapped in a container (e.g. json array).

REST architecture frameworks

- Java's restlet
  - Operators, Resources, Representations are all class entities
  - Highly pluggable implementation to support extensibility and interfaces to other web technologies such as Atom, GWT, JSON,XML,SSL,Jetty, etc..
- Frameworks in other languages include:
  - Django – in python
  - Flask – in python
  - Java Spring framework
  - Restify in Nodejs
- Few REST clients:
  - cURL
  - Postman
  - Insomnia

Tips for SENG3011

Question: How to document REST APIs?

SWAGGER

- To enable testing, all APIs and their documentation will be made available via SWAGGER
- More information on using SWAGGER
  - https://swagger.io/tools/open-source/getting-started/
  - https://idratherbewriting.com/learnapidoc/pubapis_swagger.html
  - https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api

More on SWAGGER

- Swagger Editor
  - This is the "official" text editor that can be used immediately to create documentation by hand.
  - Demo version at https://editor.swagger.io
  - Has example already populated (data saved locally in the web browser, not the cloud).
  - Can be downloaded and installed locally from https://swagger.io/tools/swagger-editor/download/
- OpenAPI Specification
  - The official specification reference currently version 3.0.2: https://swagger.io/specification/
  - Students encouraged learn the latest 3.0 version because better than v2.0 but many tools still based on 2.0

Question: How to handle input files and output files REST-fully ?

Solution #01
Enable File upload via REST commands

Input:

- Upload entire files to the web service.
  - E.G. upload pictures to Facebook, or files to Dropbox
  - Granted there are UIs to facilitate this, and for this first deliverable there is no user interface.
- Achieved through standard HTTP request verbs
  - E.G. **POST** , **PUT**
  - Make clear API(s) using HTTP for file uploads.

Solution #01
Enable File upload via REST commands

Output:

- Teams have more flexibility in module output.
- **OPTION #01**
  - Return output as JSON response.
  - This is a very common return format for API calls in the real world.
- **OPTION #02**
  - Return download links to output files.
  - Links would be returned as part of a JSON response (as opposed to all the information being contained in a JSON response as with the first option).
- **SUGGESTION** Examine the responses from API calls from available services like Twitter

Solution #02
Multipart / form-data

- Sending multipart / form-data message
- Very complicated!

Solution for SENG Workshops

- Both solutions are applicable
  - Creative, alternative and effective software designs are always impressive (to us).
- There will be points allocated to the adoption rate of your modules.
  - Practical indicator of design quality => How many people use it!
- Document Well
  - Your solution can't be used if no one knows how to use it *properly*!
- **ASK QUESTIONS!!!**
  - Filling in gaps in your knowledge and information provided : That's part of the **real process** out there
  - Asking effective questions early is paramount

Common Mistakes

- Component run accurately, but log file incomplete or doesn't exist
- No clear instructions on how to execute the component.
- The group said the version on their website is the wrong version, they will upload the correct version as soon as possible.
- Clear execution instructions, but lack of unit testing, errors generated when running the component
- Output doesn't change when changing input parameters (i.e. hardcoded the parameters)
- Who is doing what in the group, clarify from the beginning don't leave it to late.

Some References

- www.programmableweb.com
- Richardson and Ruby, RESTful Web Services by, O'Reilly, 2007 (http://oreilly.com/catalog/9780596529260)