



School of Computing and Engineering

Project Proposal Report

CP60046E

**Student Name:** Khan Najam Ul Asre

**Student ID:** 21303779

**Course:** Project – CP60046E

**Project Title:** Understanding RESTful architectural pattern by building a data-intensive web API software

**Supervisor Name:** Dr. Wei Jie

# CONTENTS

---

1	Introduction .....	3
1.1	Aim and Objectives .....	4
1.1.1	Aim .....	4
1.1.2	Objectives.....	4
2	Background .....	5
2.1	Terms Used .....	5
2.2	Problem Description .....	5
2.3	Solutions.....	6
2.3.1	SOAP Based Web Services .....	6
2.3.2	REST.....	6
2.3.2.1	Literature Gap .....	7
3	Conclusions .....	8
4	Project Plan .....	9
5	References .....	10
6	Appendix .....	11
6.1	Project Fair Form.....	11
6.2	Project Supervisor Form.....	12

# 1 INTRODUCTION

---

Over the last few decade computing technology grew at a dramatic pace. Once thought to be of the interest of large enterprises, computing technology has now become a household item. With the introduction of smart devices computing devices are now pocket items. This has changed software requirements of the business organisations. The market has become competitive. To keep going alongside the competitors, organisations have to change their marketing strategy and product presentation quickly and continuously. This requires the software system that is agile and responsive, that can be changed quickly with or without the need of redeployment. As organisation's customer base grows so does the responsiveness of the software systems. This means that software systems have to be scalable. Unfortunately, classic monolithic software systems are extremely difficult, even impossible sometimes, to be changed or scaled. This leads the software developers to favour software systems composed of small and independent components rather than the giant monolithic system. Service oriented architecture (SOA) was one answer to these problems.

SOA is an architecture which improves reusability and flexibility by building applications as components also known as services which are loosely coupled and supports interoperability (Bieberstein, Bose, Fiammante, Jones, & Shah, 2006). Microservices is a step further towards the decomposition into even smaller services. Microservices is a variant of the SOA architectural style that structures an application as a collection of loosely coupled services which are further fine-grained, modular, reusable and use lightweight protocols. There are number technology options to implement remotely distributed software systems, for example, Remote Procedure Calls (RPC) and SOAP based services. RPC lacks the standardised contracts hence needs comprehensive custom documentation. SOAP on the other hand is a heavyweight protocol and services build using SOAP are not agile enough to respond change requests quickly.

(Fielding, 2000) defined REST in his PhD dissertation "Architectural Style and Design of Network-Based Software Architecture. The REST architecture is defined by the REST constraints that affect the architectural properties like performance, scalability, simplicity, modifiability, portability and reliability.

This makes REST an ideal architecture to implement software solution that meets the before mentioned business requirements. RESTful architecture can be implemented using simple lightweight protocol. RESTful architecture is well defined by the REST constraints. It emphasises the uniform interface that is understood without or with minimal need for the documentation. It can communicate using the HTTP protocol. HTTP is not only lightweight, but is already a standard protocol for the web therefore RESTful services do not need any additional infrastructure. Ability to be hosted independently offers loose coupling and scalability. Being the popular choice for the applications of the future is the key motivation towards undertaking this project. This is a study project that is to be carried out to understand the RESTful architecture in greater detail by implementing a social task management software using this architecture.

## 1.1 AIM AND OBJECTIVES

### 1.1.1 Aim

The aim of this project is to study and understand in greater detail the concept of RESTful architectural pattern of designing web services, thereby evaluating the RESTful services compared to other technologies like SOAP services and RPC.

### 1.1.2 Objectives

1. To carry out in depth study of REST principles to understand how RESTful services differ from other options.
2. To build a data intensive web API software using REST architectural pattern to help understanding and evaluation.
3. To gain a detailed understanding of the nature of HTTP protocol and how HTTP and RESTful architecture are related to each other.
4. To make use of at least one HTTP client testing tool to gain a practical experience of how HTTP (and thereby RESTful) communication works.
5. To understand the software project life cycle.

### Implementation:

A back-end RESTful API that can be used by a mobile and/or web client that provides the functionality of task assignment and accomplishment. The idea is to create a kind of social software where there are individual users who can store their daily life TO-DOs and track and get notified when the accomplishment of such tasks is due. The functionality may include:

- a. Registration of individual user
- b. User can form one or more groups (e.g. family, university, work etc)
- c. Tasks can be registered and published to a user, or a group of users
- d. Individual users or group members can then volunteer to take responsibility of tasks, the application then notifies the change in the task status.
- e. Record task accomplishment and notify concerned members/users
- f. Allow users/group members to request others to take up the task.

**Note:** The implantation only includes the back-end API and does not include any front-end/client application.

## 2 BACKGROUND

---

### 2.1 TERMS USED

- **Monolithic system:** Systems build using traditional monolithic architecture in which system is composed as one unit where different parts of systems are heavily interdependent.
- **Microservices:** An architecture to build applications in which application is built as small independent units called services. These services are finely grained hence lightweight.
- **HTTP - Hyper Text Transfer Protocol:** A protocol used by Word wide web which define, transmit and format messages over web.
- **Interoperability:** it is the ability of computer systems which are build using different platforms and frameworks to exchange information or to work together.
- **COM - Component Object Model:** An architecture by Microsoft to build applications as components for Interoperability.
- **.Net Remoting:** A framework by Microsoft to build distributed systems.
- **Java Beans:** A framework by Java to build application as components.
- **XML - Extensible Markup Language:** It's a language which describes data which will be sent over the web or shared between different applications.
- **WSDL - Web Services Description Language:** A language written in XML which is used to define web services.
- **JAX-WS - Java API for XML:** A set of technologies by Java too build web services.
- **ASP.NET:** A web development framework to build dynamic web pages.
- **WCF - Windows Communication Foundation:** A runtime for the .NET framework which provides APIs to build serveries oriented applications.
- **API - Application Programming interface:** A mechanism which takes requests from client to server and then server responses to the client.
- **CURD:** it stands for Create, Read, Update, Delete which are very important functions for any database or for any persistent storage.
- **JSON - JavaScript Object Notation:** An open standard file format which is easy to read and easy to write for humans.
- **Loose coupling:** An approach to interconnect components in the way they are least dependent on each other.

### 2.2 PROBLEM DESCRIPTION

Since the introduction of computers continuous efforts have been made to improve software system and keep them up to date with the changing requirements. Legacy systems were large monolithic system the met the business requirements of their times. When such systems were developed, requirement would not change so rapidly. That's why the rigidity of monolithic systems was never felt. But business requirements changed over time particularly with the rapid advancement in the technology. With the technology being in access of common person changed the business environment. More businesses are able to reach the customers which resulted in the fast competition. The competition means the prompt answer to the compete, and software being at the core of the business operation has to be agile enough to respond to business requirements changes promptly.

Initially, the software systems were installed on a standalone work station. Then with the introduction of networks, the concept of client-server emerged. This required the software to be

split in server part and client part and also the infrastructure to provide communication between software parts. Initial networks were small and local area only. The organisations implemented software just within the scope of their business needs and the technology available to the organisation. Therefore, one system hardly had any similarity to the other. The systems had to communicate only within the organisation, so interoperability was not an issue.

With the success of world wide web, the business expanded themselves to reach a wider customer based. With the availability of standard web infrastructure, it became possible for businesses to collaborate and share with other business and use services from other systems. The lack of system interoperability and standardisation became a problem when it came to business collaboration. This led to the development of software frameworks that would enable organisation to design software that can communicate with software at other businesses. Different software vendors came up with their own solutions. For example, Microsoft COM component, .Net Remoting, Java Beans etc. Although systems build with a single framework were interoperable, they still lacked the interoperability between the frameworks from different vendors because they used proprietary messaging technologies.

## 2.3 SOLUTIONS

Web services were the solution to the problem. There are two approaches to web services.

### 2.3.1 SOAP Based Web Services

World Wide Web became the global infrastructure. This motivated towards the development of globally interoperable systems. Introduction of XML data format leads to the introduction of WSDL standard and SOAP bases web services. SOAP provided the global interoperability and other communication parameter. Software vendors introduced frameworks that would work with SOAP. JAX-WS, ASP.NET Web Services, WCF are few example, that were used to develop SOAP interoperable services. However, it is heavy weight protocol tightly coupled with XML. The API needed extensive documentation and protocol needed to publish the contracts. Web Services used WSDL to describe the web service and its contracts.

### 2.3.2 REST

In his PhD dissertation, (Fielding, 2000) presented the concept of REST architecture. According to him the web was iteratively developed over time with modification to the standards that defined its architecture. Therefore, web architecture was needed to guide its design, definition, and deployment. The objectives included the standardisation of interactions, scalability, loose coupling, reduced latency among others.

REST ensures these objective by imposing few constraints:

1. Client-Server: REST defines separation of concerns. Server exposes services that offer one or more capabilities. It listens for requests and responses accordingly. A client invokes those capabilities by sending request to server. Client and server have no knowledge of each other. The client communicates with server using the URI.
2. Statelessness: Communication between client and server should be stateless. This means that a request should have no knowledge of or be dependent on the previous request. Every request should contain the state information required to process it.
3. Caching: Response messages should be explicitly labelled to b cacheable or non-cacheable protection and life-time of cached information.
4. Uniform Interface: Services and consumer must share a uniform technical interface.

5. Layered System: It should be possible to introduce a layer between client and server. A layer should have no knowledge of layers beyond the immediate next layer.
6. Code on demand: This optional constraint defines that a consumer should request, and service should serve the code to process the response data.

#### **2.3.2.1 Literature Gap**

RESTful architecture itself is well documented and well defined but theoretical in nature. It defines the how the distributed system should be implemented. However, in the real systems it may not be always possible to fully adhere to a theory, REST is no exception. There might be situations in real projects where violating some of the rules may be inevitable. For example, a “resource” is at the core of REST and a URI locates the resource. However, an application may need to find the real roots of a quadratic equation. There is no concrete concept of “resource”. There is lack of literature describing such situations and some standard pragmatic approach to address this. The REST defines the resources and CRUD operations on those resources but literature describing situations beyond CRUD is rarely found.

### 3 CONCLUSIONS

---

Among two options (SOAP and REST) REST is more favourable. While SOAP is a mature protocol, it is heavy weight. SOAP is very XML centric. XML is used to describe the message. REST on the other hand can work directly over HTTP and can use the HTTP primitives to describe message. REST exploits the HTTP not only for the message transmission but also for communicating response status and content negotiation. This makes REST more flexible. The statelessness opens up the doors for scalability. Statelessness requests to be independent of each other which facilitates load balancing. Since a request does not need any information about other request, it is possible that requests related to one large operations can be sent to different servers. If built on HTTP, REST services can be deployed and hosted on any http server. The cache-ability significantly reduces the server traffic and reduces the latency. Although SOAP is a well-defined protocol, it does not dictate on the interface design. It relies on the contract publishing rather than uniform interface. This requires extensive documentation. REST relies on the uniform interface, so if designed properly, interface is predictable.

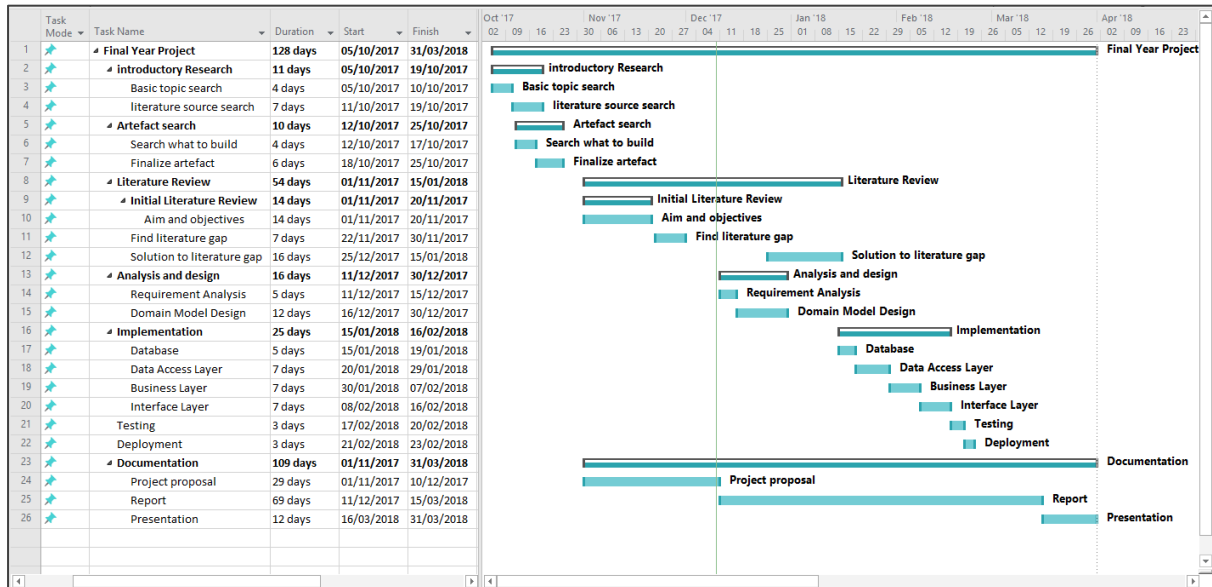
Compared to SOAP which is very XML centric, REST does not rely on specific data format. With HTTP content negotiation it can advise server of data format of request message and can request a particular data format for the response. A key benefit is that it can work with any data format with the same efficiency. This makes it convenient to use JSON as the data format. JavaScript is very good at parsing the JSON format. Using JSON as data format means that it is JavaScript friendly. Almost every browser supports JavaScript. Even JavaScript is gaining popularity on the server side with frameworks like NodeJS. Therefore, REST can target wide range of API clients.

For the reasons above, this project intends to architect software product on REST principles which will require the detailed study and understanding of REST principles. The proposed software is data intensive which means it will require intensive client-server communication. More data means more operations. This will help making use of most of the REST's capabilities. With popularity of web in business and public domain, having knowledge of HTTP has become almost essential of any software developer. Understanding of HTTP means ability to work with wide variety of systems and devices.



## 4 PROJECT PLAN

Gantt chart is used for project plan.



## 5 REFERENCES

---

- Bieberstein, N., Bose, S., Fiammante, M., Jones, K., & Shah, R. (2006). *Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap (The developerWorks Series)*. Westford, Massachusetts, United States of America: IBM Press.
- Fielding, R. T. (2000). *Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)*. Retrieved from [www.ics.uci.edu](http://www.ics.uci.edu):  
[https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
- Richardson, L., & Ruby, S. (2007). *RESTful Web Services* (1st ed.). United States of America: O'Reilly Media.
- Severance, C. (2015, 6 11). *Roy T. Fielding: Understanding the REST Style - IEEE Journals & Magazine*. Retrieved from [ieeexplore.ieee.org](http://ieeexplore.ieee.org):  
<http://ieeexplore.ieee.org.ezproxy.uwl.ac.uk/document/7122189/>
- Webber, J., Parastatidis, S., & Robinson, I. (2010). *REST in Practice: Hypermedia and Systems Architecture* (1st ed.). United States of America: O'Reilly Media.

## 6 APPENDIX

### 6.1 PROJECT FAIR FORM



School of Computing and Engineering

#### Project Fair Form

Module Name	Project – Level 6
Student Name	KHAN NAJAM UL ASRE
Student ID	21303779

Project Supervisor name	Research Topic discussed	Sign by the supervisor
Dr Wei Jie	Design and implementation of webservices.	
Dr Elahe Kani Zabihi	Online Data mangement System. <i>(Antoine khurikha)</i>	
Dr Nasser Matkorian	Progressive web Application Accessibility	

Please note: It is expected of you to feedback comments received from at least 3 supervisors

## 6.2 PROJECT SUPERVISOR FORM



**University of West London**

**School of Computing and Engineering**

**PROJECT MODULE**

**Module Code**

CP60046E

**Project Supervisor Form**

**Student Ref. No.** 21303779

**Submission Date** 26/10/17 **Deadline Time**

**Student's Surname.** ASre

**Student's Forename.** Khan Najam VL

**Project Supervisor's Name.** Dr Wei Jie


**Project topic**

Development of RESTful API: Exploration and understanding of RESTful principles and architectural constraints by the development of

**Comment by the supervisor**

a comprehensive back-end API for a task assignment network software App.

This is a solid proposal with sufficient technical development. It is appropriate for a V6 project.

	Signatures	Date
<b>Student</b>	Najam	26/10/2017
<b>Supervisor</b>		26/10/2017