Informatics Institute of Technology

In collaboration with

University of Westminster, UK.

“**Visual Dataflow Programming based Microservice Platform”**

A dissertation by

Sajith Dilshan Jamal

Supervised by

Sriyal Himesh Jayasinghe

Submitted in partial fulfilment of the requirements for the

M.Sc in Advanced Software Engineering

**August 2017**

©The copyright for this project and all its associated products resides with Informatics Institute of Technology

|  |  |
| --- | --- |
| Declaration I hereby certify that this project report and all the artefacts associated with it is my **own** work and it has not been submitted before nor is currently being submitted for any degree programme.  Full Name of Student: Sajith Dilshan Jamal  Registration No : 2015014 | |
| Signature : ................................................... | Date : ........................................ |

# Abstract

Up until now the traditional way of developing software has been to take the monolithic approach is some form; for example *‘Service Oriented Architecture’* or *’Enterprise Applications’.* However due to its weaknesses, a newer approach called *‘Microservices’* has been recently gaining more traction. However microservices are still in its infancy. Due to this fact microservices comes with its own host of problems such as inter service communication, unwieldy IDEs, inadequate testing mechanisms.

The Dataflow programming based Microservice Platform is a system which combines the concepts of dataflow programming and the strengths of Microservices to build a platform which addresses these shortcomings.

The platform was developed using NodeJS as the server-side language, AngularJS as the front end language and MongoDB as the database. Implemented system was tested thoroughly under different conditions and the prototype system was evaluated by evaluators of various domains. Eventually, the test results attested that the analysis, design, implementation and documentation have been carried out in an effective and in an efficient manner.

**Subject Descriptors**

C.2.1 Network Architecture and Design  
 D.1.7 Visual Programming

**Keywords**

Microservices, Dataflow Programming, Platform, NodeJS

# Acknowledgement

The effort that was put by me in to completing this project has been a very tough expedition throughout but at the same time it was a wonderful experience. However, it would not have been possible without the kind support, assistance and courage of many people who stood behind me. I would like to extend my sincere thanks to all of them.

I am highly indebted to my supervisor Mr Sriyal Himesh Jayasinghe, Informatics Institute of Technology for his constant guidance and supervision as well as for encouragement and ideas given me to strive to greater heights and to achieve my fullest potential.

I would like to express my special gratitude and thanks to everyone who assisted me by participating online surveys, interviews and contributing their ideas for requirements gathering and critical evaluation.

# Contents

[Declaration I](#_Toc489192537)

[Abstract II](#_Toc489192538)

[Acknowledgement III](#_Toc489192539)

[Contents IV](#_Toc489192540)

[List of Figures X](#_Toc489192541)

[List of Tables XI](#_Toc489192542)

[Table of Equations XII](#_Toc489192543)

[List of Abbreviations XIII](#_Toc489192544)

[1. Introduction I](#_Toc489192545)

[1.1 Chapter Overview 1](#_Toc489192546)

[1.2 Project Background 1](#_Toc489192547)

[1.3 Problem Domain 1](#_Toc489192548)

[1.3.1 Resulting context 1](#_Toc489192549)

[1.3.2 Drawbacks of using microservices 2](#_Toc489192550)

[1.3.3 Problem domain focus 2](#_Toc489192551)

[1.4 Dataflow Programming Paradigm 3](#_Toc489192552)

[1.5 Previous Work 3](#_Toc489192557)

[1.6 Microservice Platform 3](#_Toc489192558)

[1.7 Project Title 4](#_Toc489192559)

[1.8 Aim and Project Objectives 4](#_Toc489192560)

[1.9 Proposed Framework 4](#_Toc489192561)

[1.10 Primary Objectives 5](#_Toc489192562)

[1.11 Development Resource Requirements 7](#_Toc489192563)

[1.12 Project deliverables 7](#_Toc489192564)

[1.12.1 Functional artefact 7](#_Toc489192565)

[1.12.2 Documentation 7](#_Toc489192566)

[1.12.3 Knowledge 7](#_Toc489192567)

[2. Literature Review I](#_Toc489192568)

[2.1 Chapter Overview 8](#_Toc489192569)

[2.2 Microservices 8](#_Toc489192570)

[2.2.1 Architecture 8](#_Toc489192571)

[2. 2.2.1.1 Platform Architecture 8](#_Toc489192572)

[2.2.1.2 Platform Support Architecture 12](#_Toc489192573)

[2.2.1.3 Communication 15](#_Toc489192574)

[2.2.2 Hardware Architecture 17](#_Toc489192575)

[2.2.3 Products 17](#_Toc489192576)

[2.2.3.1 Docker 17](#_Toc489192577)

[2.3 Dataflow Programming 20](#_Toc489192578)

[2.3.1 Products 20](#_Toc489192579)

[2.3 Chapter Summary 22](#_Toc489192582)

[3. Project Management 26](#_Toc489192583)

[3.1 Chapter Overview 24](#_Toc489192584)

[3.2 Project Management Methodology 24](#_Toc489192585)

[3.3 Time allocation 24](#_Toc489192586)

[3.4 Constraints and Dependencies 25](#_Toc489192587)

[3.5 Potential Risks and Mitigation Plan 25](#_Toc489192588)

[3.6 Development Methodology 26](#_Toc489192589)

[3.7 Research Methodology 28](#_Toc489192590)

[3.8 Chapter Summary 28](#_Toc489192591)

[4. System Requirements Specification 9](#_Toc489192592)

[4.1 Chapter Overview 29](#_Toc489192593)

[4.2 Requirements Elicitation Process 29](#_Toc489192594)

[4.2.1 End user questionnaire 30](#_Toc489192595)

[4.2.1.1 Structure of the questionnaire 30](#_Toc489192596)

[4.2.1.2 Limitations of the Questionnaire Process 31](#_Toc489192597)

[4.2.1.3 Requirement elicitation of questionnaire 31](#_Toc489192598)

[4.2.2 Formal Interviews 37](#_Toc489192599)

[4.2.2.1 Structure of the interview questions 37](#_Toc489192600)

[4.2.2.2 Limitations of the interview process 37](#_Toc489192601)

[4.2.3 Findings derived from requirements elicitation process 38](#_Toc489192602)

[4.3 Stakeholders 38](#_Toc489192603)

[4.3.1 Stakeholders Onion Diagram 38](#_Toc489192604)

[4.3.2 Roles 39](#_Toc489192605)

[4.4 Context Diagram 40](#_Toc489192613)

[4.5 Use Case Diagram 41](#_Toc489192614)

[4.6 Use Cases 42](#_Toc489192615)

[4.7 Functional Requirements 44](#_Toc489192625)

[4.8. Non Functional Requirements 45](#_Toc489192636)

[4.9. Chapter Summary 46](#_Toc489192637)

[5. System Design 25](#_Toc489192638)

[5.1 Chapter Overview 47](#_Toc489192643)

[5.2 High Level Design 47](#_Toc489192645)

[5.2.1. Rich Picture of the Microservice Platform 47](#_Toc489192647)

[5.2.2. High Level Architecture 48](#_Toc489192648)

[5.3 System Design 50](#_Toc489192649)

[5.2.3. Selection of Design Methodology 50](#_Toc489192650)

[5.4 Domain Model of Microservice Platform 51](#_Toc489192658)

[5.2.4. Data Transfer Objects 52](#_Toc489192659)

[5.5. Class Diagram 52](#_Toc489192665)

[5.6. Sequence Diagram 53](#_Toc489192666)

[5.7. Dataflow Tool 58](#_Toc489192667)

[5.7.1. Dataflow Tool Components 59](#_Toc489192668)

[5.7.1.1. Begin Component 59](#_Toc489192669)

[5.7.1.2. Database Query Component 60](#_Toc489192670)

[5.7.1.3. Script Component 60](#_Toc489192671)

[5.7.1.4. Decision Component 60](#_Toc489192672)

[5.7.1.5. Database Insert Component 60](#_Toc489192673)

[5.7.1.6. REST Call Component 60](#_Toc489192674)

[5.7.1.7. Microservice Component 60](#_Toc489192675)

[5.7.1.8. End Component 60](#_Toc489192676)

[5.8. Microservice Platform 61](#_Toc489192677)

[5.8.1. Docker Container 61](#_Toc489192678)

[5.8.2. Docker Deployer and Microservice Manager 61](#_Toc489192679)

[5.8.3. Microservice Compiler 62](#_Toc489192680)

[5.8.4. Service Registry 62](#_Toc489192681)

[5.8.5. Microservice Platform 62](#_Toc489192682)

[5.9. Chapter Summary 63](#_Toc489192683)

[6. System Implementation 43](#_Toc489192684)

[6.1. Chapter Overview 64](#_Toc489192686)

[6.2. Programming Languages 64](#_Toc489192687)

[6.2.1. Javascript 64](#_Toc489192688)

[6.2.2. XML 64](#_Toc489192689)

[6.2.3. JSON 65](#_Toc489192690)

[6.3. Choice of IDEs 65](#_Toc489192691)

[6.4. Coding Standard 66](#_Toc489192692)

[6.4.1. Variable and Method Naming 66](#_Toc489192693)

[6.4.2. Class Names 66](#_Toc489192694)

[6.4.3. Comments 66](#_Toc489192695)

[6.5. External Libraries 66](#_Toc489192696)

[6.5.1. NodeJS 66](#_Toc489192697)

[6.5.2. AngularJS 67](#_Toc489192698)

[6.5.3. mxGraph 67](#_Toc489192699)

[6.6. Prototype Features 67](#_Toc489192700)

[6.6.1. Dataflow tool 67](#_Toc489192701)

[6.6.1.1. Dataflow tool Output 68](#_Toc489192702)

[6.6.2. Microservice Platform 69](#_Toc489192703)

[6.7. Chapter Summary 71](#_Toc489192704)

[7. System Testing 54](#_Toc489192705)

[7.1. Chapter Overview 72](#_Toc489192707)

[7.2. Objectives and goals for Testing 72](#_Toc489192708)

[7.3. Testing Process 72](#_Toc489192709)

[7.4. Testing Criteria 73](#_Toc489192710)

[7.5. Adopted testing method for functional requirement 74](#_Toc489192711)

[7.6. Areas Tested – Functional Requirements 74](#_Toc489192712)

[7.6.1. Dataflow Tool and Microservice Platform 74](#_Toc489192713)

[7.6.2. Dataflow Tool components, DataTransferObjects and Transpiler 76](#_Toc489192714)

[7.6.3. Swagger Output Testing 78](#_Toc489192715)

[7.6.4. Service Registry Testing 78](#_Toc489192716)

[7.6.5. Docker Container Testing 78](#_Toc489192717)

[7.6.6. Functional Requirements Met 78](#_Toc489192718)

[7.7. Areas Tested – Non Functional 78](#_Toc489192719)

[7.8. Chapter Summary 80](#_Toc489192720)

[8. Evaluation 54](#_Toc489192721)

[8.1. Chapter Overview 81](#_Toc489192723)

[8.2. Evaluation Criteria 81](#_Toc489192724)

[8.3. Selection of Evaluators 81](#_Toc489192725)

[8.4. Evaluation Methodology and Approach 82](#_Toc489192726)

[8.5. Summary of Evaluation of Survey Questions 82](#_Toc489192727)

[8.6. Evaluation of Survey Results 83](#_Toc489192728)

[8.6.1. Overall Concept 83](#_Toc489192729)

[8.6.2. Scope & depth of the project 85](#_Toc489192730)

[8.6.3. System Design, Architecture and Implementation 85](#_Toc489192731)

[8.6.4. Solution and Prototype 86](#_Toc489192732)

[8.6.5. Usability Accuracy and Performance 87](#_Toc489192733)

[8.6.6. Limitations of the solution and future enhancements 87](#_Toc489192734)

[8.7. Self Evaluation 88](#_Toc489192735)

[8.7.1. Self evaluation on identified evaluation criteria 88](#_Toc489192736)

[8.7.2. Reflection on the Functional Requirements 89](#_Toc489192737)

[8.8 Chapter Summary 90](#_Toc489192745)

[9. Conclusion 54](#_Toc489192746)

[9.1. Chapter Overview 91](#_Toc489192748)

[9.2. Achievement of Aim and Objectives 91](#_Toc489192749)

[9.3. Utilizing of Knowledge from Course Modules 93](#_Toc489192750)

[9.4. Use of existing skills 94](#_Toc489192751)

[9.5. Learning outcomes 94](#_Toc489192752)

[9.6. Problems and Challenges Faced 95](#_Toc489192753)

[9.7. Limitations of the Research 95](#_Toc489192754)

[9.8. Future Enhancements 96](#_Toc489192755)

[9.9. Contributing Remarks 97](#_Toc489192756)

[9.10. Concluding Remarks 97](#_Toc489192757)

[Appendix A 1](#_Toc489192758)

[Appendix B 1](#_Toc489192759)

[1.1 Dataflow Tool Testing 1](#_Toc489192760)

[1.2 Begin Component Testing 2](#_Toc489192761)

[1.2.1 DataTransferObject Testing 2](#_Toc489192762)

[1.2.2 Input Object Testing 2](#_Toc489192763)

[1.3 Database Query Component Testing 3](#_Toc489192764)

[1.4 Script Component Testing 3](#_Toc489192765)

[1.5 Decision Component Testing 4](#_Toc489192766)

[1.6 Database Insert Component Testing 7](#_Toc489192767)

[1.7 REST Component Testing 7](#_Toc489192768)

[1.8 End Component Testing 7](#_Toc489192769)

[Appendix C 8](#_Toc489192770)

[References 1](#_Toc489192771)

# List of Figures

[Figure 1.9‑1 Overview of proposed Framework 4](#_Toc489188472)

[Figure 2.2‑1 Example diagram of BPMN Model(Bpmn-io n.d.) 11](#_Toc489188473)

[Figure 2.2‑2 Docker Capabilities(Docker Inc n.d.) 18](#_Toc489188474)

[Figure 2.2‑3 Tool used for developing Microsoft Visual Programming Programs(Microsoft n.d.) 21](#_Toc489188475)

[Figure 2.2‑4 Developing in Nextflow(Nextflow n.d.) 22](#_Toc489188476)

[Figure 2.2‑5 Developing in Openwire(Openwire n.d.) 23](#_Toc489188477)

[Figure 4.2‑1 Preferred Hosting Solutions 33](#_Toc489188478)

[Figure 4.2‑2 Preferred Deployment Solutions 33](#_Toc489188479)

[Figure 4.2‑3 Preferred Languages 34](#_Toc489188480)

[Figure 4.2‑4 Preferred IDEs 34](#_Toc489188481)

[Figure 4.2‑5 Preferred communication method 35](#_Toc489188482)

[Figure 4.2‑6 Preferred Testing Solutions 36](#_Toc489188483)

[Figure 4.2‑7 Preferred Testing Method 36](#_Toc489188484)

[Figure 4.2‑8 Preferred CI Strategy 37](#_Toc489188485)

[Figure 4.2‑9 Preferred Environment Solution 37](#_Toc489188486)

[Figure 4.2‑10 Whether or not multiple versions run 38](#_Toc489188487)

[Figure 4.2‑11 Onion Diagram 39](#_Toc489188488)

[Figure 4.2‑12 Context Diagram 41](#_Toc489188489)

[Figure 4.2‑13 Use Case Diagram 42](#_Toc489188490)

[Figure 5.2‑1 Rich Picture 49](#_Toc489188491)

[Figure 5.2‑2 High Level Architecture Diagram 50](#_Toc489188492)

[Figure 5.4‑1 Domain Model of Microservice Platform 52](#_Toc489188493)

[Figure 5.5‑1 Class Diagram 53](file:///E:\IIT%20MSC\Dissertation\gdrive\IIT%20MSc\Dissertation\dissertation_v12.docx#_Toc489188494)

[Figure 5.6‑1 Sequence diagram for login 54](#_Toc489188495)

[Figure 5.6‑2 Get Projects 55](#_Toc489188496)

[Figure 5.6‑3 Get Environments 55](#_Toc489188497)

[Figure 5.6‑4 Sequence Diagram for Add Environment 56](#_Toc489188498)

[Figure 5.6‑5 Sequence Diagram of Add Project 56](#_Toc489188499)

[Figure 5.6‑6 Sequence Diagram of Creating a dataflow 57](#_Toc489188500)

[Figure 5.6‑7 Save Dataflow 58](#_Toc489188501)

[Figure 5.6‑8 Deploy Microservice 58](#_Toc489188502)

[Figure 5.7‑1 Dataflow Mockup 59](#_Toc489188503)

[Figure 5.8‑1 Component Level View of Microservice Platform 62](#_Toc489188504)

[Figure 5.8‑2 Dockerfile Screenshot 63](#_Toc489188505)

[Figure 6.6‑1 Dataflow tool screenshot 69](#_Toc489188506)

[Figure 6.6‑2 Template App Folder Structure 70](#_Toc489188507)

[Figure 6.6‑3 OS Commands 71](#_Toc489188508)

[Figure 6.6‑4 Template App Logic Location 71](#_Toc489188509)

[Figure 6.6‑5 Template App Docker 71](#_Toc489188510)

[Figure 6.6‑6 *“docker images”* result 72](#_Toc489188511)

[Figure 6.6‑7 “*docker ps”* result 72](#_Toc489188512)

[Figure 7.3‑1 Component Testing Flow 74](#_Toc489188513)

[Figure 7.7‑1 Results of how long developers took to familiarize with the system 80](#_Toc489188514)

# List of Tables

[Table 1.10‑1 Table of Objectives 7](#_Toc489188515)

[Table 2.2‑1 Containers Review 10](#_Toc489188516)

[Table 2.2‑2 BPMN Review 12](#_Toc489188517)

[Table 2.2‑3 Versioning Review 12](#_Toc489188518)

[Table 2.2‑4 Platform Architecture Research Paper Summary 13](#_Toc489188519)

[Table 2.2‑5 Domain Driven Design Review 14](#_Toc489188520)

[Table 2.2‑6 Continuous Delivery Review 15](#_Toc489188521)

[Table 2.2‑7 Ports and Adapters Pattern Review 15](#_Toc489188522)

[Table 2.2‑8 Platform Support Architecture Research Paper Summary 16](#_Toc489188523)

[Table 4.3‑1 Gantt Chart 26](#_Toc489188524)

[Table 4.6‑1 Development Methodologies 28](#_Toc489188525)

[Table 3.2‑1 Summary of Requirements Gathering techniques 31](#_Toc489188526)

[Table 3.2‑2 Limitations of Questionairres 32](#_Toc489188527)

[Table 3.2‑3 Limitations of using interviews 39](#_Toc489188528)

[Table 5.4‑1 Overview of domain model classes 52](#_Toc489188529)

[Table 4.5‑1 Class Diagram Explanation 54](#_Toc489188530)

[Table 6.6‑1 Dataflow Tool Testing Result Summary 76](#_Toc489188531)

[Table 6.6‑2 2016 Browser usage statistics 77](#_Toc489188532)

[Table 6.6‑3 2017 Browser Usage statistics 77](#_Toc489188533)

[Table 6.6‑4 Componenet Test Case Handling 78](#_Toc489188534)

[Table 6.6‑5 Functional Requirements Testcases 79](#_Toc489188535)

[Table 7.2‑1 Evaluation Criteria 82](#_Toc489188536)

[Table 7.3‑1 Evaluators 83](#_Toc489188537)

[Table 7.5‑1 Summary of questions 84](#_Toc489188538)

[Table 7.6‑1 Feedback Evaluation – Overall Concept 85](#_Toc489188539)

[Table 7.6‑2 Feedback Evaluation – Scope and Depth 86](#_Toc489188540)

[Table 7.6‑3 Feedback Evaluation - System Design, Architecture and Implementation 87](#_Toc489188541)

[Table 7.6‑4 Feedback Evauation - Solution and Prototype 88](#_Toc489188542)

[Table 7.7‑1 Self Evaluation 89](#_Toc489188543)

[Table 7.7‑2 Reflection of Functional Requirements 90](#_Toc489188544)

[Table 8.8‑1 Future Enhancements 98](#_Toc489188545)

# Table of Equations

[Equation 4.2‑1 Response Rate of Questionnaire 32](#_Toc489188548)

[Equation 7.6‑1 Pass rate calculation formula 76](#_Toc489188549)

[Equation 7.6‑2 Pass rate calculation formula without IE 76](#_Toc489188550)

# List of Abbreviations

|  |  |
| --- | --- |
| **Abbreviation** | **Definition** |
| **SOA** | Service Oriented Architecture |
| **IE** | Internet Explorer |
| **REST** | Representational State Transfer |
| **HTTP** | Hyper Text Transfer Protocol |
| **AMQP** | Advanced Message Queuing Protocol |
| **IDE** | Integrate Development Environment |
| **JVM** | Java Virtual Environment |
| **AWS** | Amazon Web Services |
| **GAE** | Google App Engine |
| **OOADM** | Object-Oriented Analysis and Design Method |
| **SSADM** | Structured Systems Analysis and Design Method |
| **REST** | Representational State Transfer |
| **SRS** | Software Requirement Specification |
| **JSON** | JavaScript Object Notation |
| **UML** | Unified Modelling Language |
| **DDD** | Domain Driven Design |

# Introduction

Contents

* Chapter Overview
* Project Background
* Problem Domain
* Dataflow Programming Paradigm
* Microservice Platform
* Project Title
* Aims and Objectives
* Proposed Framework
* Primary Objectives
* Main Components of the Framework
* Design Philosophy and Project Management Methodology
* Development Resource Requirements
* Project Schedule
* Project Deliverables

# Chapter Overview

The purpose of this chapter is to provide an overview of the project undertaken. It starts by outlining a brief introduction along with the current limitations faced by microservice developers. Additionally previous work done in relation to designing and developing microservices is evaluated along with the aim and objectives that were laid out for the successful completion of the project are also included. Furthermore the high level features of the system are discussed and the final project deliverables are mentioned.

# Project Background

In software engineering, a monolithic application can describe a single/multi tier architecture program in which the user interfaces, data access objects and business logic are combined into a single program from a single platform. They are self contained, and independent from other computing applications. The design philosophies for monolithic applications is such that the application is responsible not just for a particular task, but can perform every step needed to complete a particular function. The weaknesses of monolithic application design is discussed here(Richardson n.d.).

# Problem Domain

One of the proposed approaches to address the shortcomings of monolithic design philosophies is an approach of SOA(Brown 2008; Douglas n.d.) called Microservices (Newman 2015; Casterson 2016)Designers architect the application by applying the Scale Cube (Abbott & Fisher 2009) (specifically y-axis scaling) and functionally decompose the application into a set of collaborating services. Each service implements a set of narrowly, related functions. Services communicate using either synchronous protocols such as HTTP/REST or asynchronous protocols such as AMQP. Services are developed and deployed independently of one another. Each service has its own database in order to be decoupled from other services. When necessary, consistency is between databases is maintained using either database replication mechanisms or application-level events.

## Resulting context

This solution has a number of benefits:

* Each microservice is relatively small
  + Easier for a developer to understand
  + The IDE is faster making developers more productive
  + The web container starts faster, which makes developers more productive, and speeds up deployments
* Each service can be deployed independently of other services - easier to deploy new versions of services frequently
* Easier to scale development - allows to organize the development effort around multiple teams. Each team is responsible a single service. Each team can develop, deploy and scale their service independently of all of the other teams.
* Improved fault isolation.
* Each service can be developed and deployed independently
* Eliminates any long-term commitment to a technology stack

## Drawbacks of using microservices

* Developers must deal with the additional complexity of creating a distributed system.
  + Developer tools/IDEs are oriented on building monolithic applications and don’t provide explicit support for developing distributed applications.
  + Testing is more difficult
  + Developers must implement the inter-service communication mechanism.
  + Implementing use cases that span multiple services without using distributed transactions is difficult
  + Implementing use cases that span multiple services requires careful coordination between the teams
* Deployment complexity. In production, there is also the operational complexity of deploying and managing a system comprised of many different service types.
* Increased memory consumption. The microservices architecture replaces N monolithic application instances with NxM services instances. If each service runs in its own JVM (or equivalent), which is usually necessary to isolate the instances, then there is the overhead of M times as many JVM runtimes. Moreover, if each service runs on its own VM (e.g. EC2 instance), as is the case at Netflix, the overhead is even higher.

## Problem domain focus

While the management of memory consumption can be written as a separate thesis itself, this research will focus on trying to solve the first two drawbacks of using microservices by implementing a dataflow language (Kent 2000) layer on top of a microservice platform layer.

# Dataflow Programming Paradigm

In computer programming, dataflow programming models business cases as directed graphs of the data flowing between operations thereby implementing dataflow principles.

Traditionally a program is modeled into a series of operations happening in a specific order (sequential, procedural, control flow, imperative programming), focusing on commands in line with von Neuman’s vision of dataflow programming (Johnston et al. 2004) where the data remains “at rest”. In contrast, dataflow programming emphasizes the movement of data from one state/operation to another, and models programs as a series of connections where explicitly defined inputs and outputs connect one operation to another which function like black boxes (of Technology. Automatic Programming Information Centre 1972)



# Previous Work

While no standardization of any form yet exists there are a myriad of case studies (Gadea et al. 2016; Heorhiadi et al. 2016; Ueda et al. 2016; Linthicum 2016) available. These case studies will be analysed for their strengths and weaknesses and their best features used in the design of the project.

Also available are attempts at productization of microservices by monoliths in the Cloud Hosting Domain like Amazon Web Services(Stroppa 2016) and Google(Google Inc n.d.). These products are however bound to the platforms they rely on and cannot be truly open.

None of the above mentioned have actually tried to solve the mentioned problems of developing microservices, therefore this research will focus approaching the problem by a different angle: using a dataflow based approach to make development easier for microservices and build a microservice architecture using a platform, thereby alleviating developer overhead of needing to build a microservice architecture as part of the solution to building their product. This platform will be flexible and generic enough to apply to any use case possible.

# Microservice Platform

In order to solve the second drawback this research intends to focus in creating a microservice management platform. However microservices are still an emergent architecture in its infancy, therefore there is no standard yet that exists that describes the required components and design principles needed to build a microservice or its support architecture. Instead there exists a myriad of research papers discussing case studies of the implementation of microservices. Using these case studies as a base this research intends to evaluate components and architectural principles used, in order to propose a standard for creating microservice architecture, and its management platforms.

# Project Title

A dataflow programming based microservices platform

# Aim and Project Objectives

*Design, develop and evaluate a framework that will help web developers build microservice projects using a dataflow programming language*

To elaborate further, this research will focus on two sections:

* Dataflow programming
* Microservice Platform

A visual dataflow language will be created to be used by developers to model their microservice in. This dataflow model will then be compiled into a microservice and deployed for the developers to use.

This research will not focus on:

* Trying to import existing microservices into the platform in any form
* Research in eliminating drawbacks in tools used by developers of microservices. Instead a new approach for developing microservices is proposed in the form of the dataflow language.

# Proposed Framework

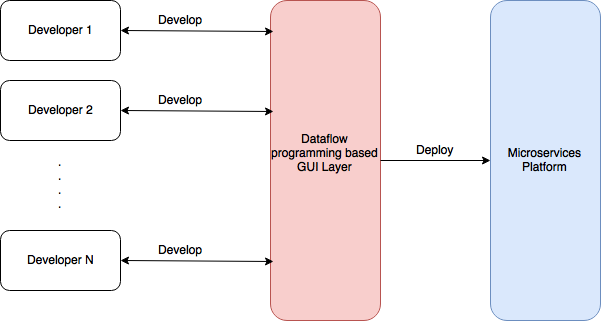


Figure ‑ Overview of proposed Framework

# Primary Objectives

|  |  |
| --- | --- |
| Objective 1 | Prepare Terms of Reference |
| Prepare the Terms of Reference which defines the aims and objectives, features of the prototype, background, resource requirements, project deliverables and activity schedule. | |
| Objective 2 | Literature Survey |
| Carry Out an in-depth literature survey on the following subject areas   * Microservices: paradigm, philosophy, design approaches, existing products and implementation case studies * Dataflow languages: existing languages and frameworks | |
| Objective 3 | Selection of a software development methodology |
| Select a software development methodology that would be most suitable to carry out the various phases of the project | |
| Objective 4 | Requirement gathering process |
| Carryout an in-depth user requirement gathering phase with:   * Developers who use microservice frameworks to develop applications * Architects who design applications around microservice frameworks * Project managers who manage teams that develop applications using microservice frameworks | |
| Objective 5 | Prepare a software requirement specification (SRS) |
| Using the information gathered through the literature review, end user questionnaire and observations, domain expert interviews and personal evaluations prepare the software requirements specification to document the functional and non-functional requirements of the proposed system | |
| Objective 6 | Selection of software and hardware resources |
| Select the most appropriate technologies, tools, APIs, libraries, platforms, algorithms and hardware requirements to implement the prototype | |
| Objective 7 | Prepare software design specification |
| Prepare the design specification for the prototype according to the analyzed requirements gathered from the requirement gathering phase and selected software and hardware resources | |
| Objective 8 | Develop the prototype |
| Develop the prototype using the most appropriate software and hardware resources to full fill the user requirements identified in the requirement specification. | |
| Objective 9 | Testing of the prototype |
| Formulate a testing plan, prepare test cases and conduct an in-depth testing of the system to identify bugs and check whether the required functional and non-functional requirements of the users are achieved from the developed prototype | |
| Objective 10 | Evaluation of the work carried out |
| * Carry out a critical evaluation of the prototype using selected user groups of the system and conduct a review of the evaluation findings to determine how far the project has successfully addressed the hypothesis * Carry out a review with domain experts of different research areas used in the prototype to identify areas for future improvements * Perform a personal evaluation to self asses the work carried out | |
| Objective 11 | Documentation |
| Document all findings and key steps involved in the project and submit the project report | |

Table ‑ Table of Objectives

# Development Resource Requirements

* A pc/laptop (preferred minimum requirements: core i3 processor, 4GB DDR3 RAM, 100GB hard drive)
* Ubuntu installed on said machine
* Development IDE (preferably on online development environment like <https://c9.io/> will be used, however in case of complications ‘Eclipse Neon for Java EE developers’ will be used instead)
* Internet Connectivity
* NodeJS and NPM installed on development Machine

# Project deliverables

## Functional artefact

A working model of the proposed framework will be available for demonstration, evaluation and future enhancements.

## Documentation

Project proposal, project poster, interim project report, draft project report, and the final project report consisting of literature survey, design, UML diagrams.

## Knowledge

Knowledge gained by performing the research.

# Literature Review

Contents

* Chapter Overview
* Microservices Architecture
* Microservices Products
* Dataflow Programming
* Chapter Summary

# Chapter Overview

The purpose of this chapter is to present the reader with a literature review of microservice and dataflow programming products, libraries, languages, and case studies of implementations.

The scope of this chapter is to investigate existing technologies related to microservices and dataflow programming to provide the reader with a review of each, researching advantages and weaknesses of each with the intention of using components derived from them.

# Microservices

Microservices being a design principle like SOA (Douglas n.d.) rather than an actual library, as well as being a principle still coming out of its infancy stages, does not yet have many successful attempts at standardizing its implementations(Sill 2016; Newman 2015) like SOA(Brown 2008) does. This was inferred by there not only not having any authorities, books, or standards available. There also exists many case studies of implementations and design patterns as well as little known attempts at products that are highly dependent on running on the architecture and technologies of the company that developed it(Microsoft n.d.; WSO2 n.d.). Also due to it being a principle by nature, many existing products have been repurposed and rebranded to run as microservice products. The most famous of this category being Docker(Docker Inc n.d.), followed by AWS (Amazon Inc n.d.) and Google App Engine(Google Inc n.d.).

These implementations, case studies and products will be reviewed with the intention of using principle components to design a microservice platform.

## Architecture

As discussed during the introduction, microservices are still an emergent architecture in its infancy, therefore there is no standard yet that exists that describes the required components and design principles needed to build a microservice or its support architecture. Instead there exists a myriad of research papers discussing case studies of the implementation of microservice. In order to build the architecture of the microservice platform, these case studies will be reviewed to find the most desirable architectural components to be implemented into the microservices platform.

### 2.2.1.1 Platform Architecture

#### Containers

The first and foremost point when discussing microservices is the separation of functionality into separate pieces. It is how and where these pieces are deployed and hosted that is of concern to us. The most successful solution is the usage of the concept of containerization (Stubbs et al. 2015; Fetzer 2016; Heorhiadi et al. 2016; Amaral et al. 2015).

Operating System level virtualization (containerization) is a server virtualization method where the kernel of the operating system allows the existence of multiple isolated user space instances instead of just one. Such instances, sometimes called containers, virtualization engines, or jails, may look and feel like a real server from the point of view of the running application.

|  |  |
| --- | --- |
| Advantages Identified | Disadvantages Identified |
| * Scalability * Elasticity * Clusterability * On demand Instantiation * Isolation of execution environment * Rapid application deployment * Portability * Component reuse * Version control * Minimal overhead * Lightweight footprint * Simplified maintenance | * Lack of a communication system * Lack of inbuilt security |
| Impact:  Separation of functionality into discrete sets is an ingrained requirement of being able to build microservice architectures. This is supported by the fact that so many papers reference this principle if not in name but by at least its concept. Its disadvantages are an architectural problem to be solved and its advantages make it a highly sought after feature. | |

Table ‑ Containers Review

#### Front End GUI

Several papers as shown in the able table discuss the requirement of Front end GUIs. However building UIs that can meet viable standards(Zimmermann 2007; Abbasi et al. 2012) is out of scope of this project, and so will not be attempted.

#### Business Process Model and Notation(BPMN)

BPMN(BPMN n.d.) refers to a structural representation, description or diagram which defines a specified flow of activities in a particular business or organizational unit.BPMN modeling is not dissimilar to what dataflow programming is trying to accomplish.

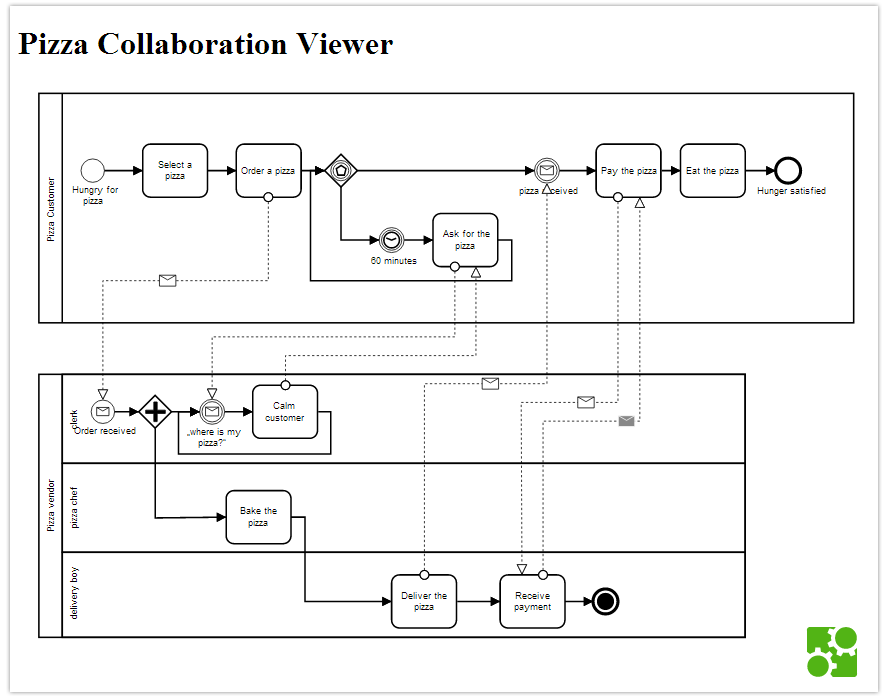


Figure ‑ Example diagram of BPMN Model(Bpmn-io n.d.)

|  |  |
| --- | --- |
| Advantages Identified | Disadvantages Identified |
| * Manage complexity * Automate generation of a real working system or part of a system from models * Automate repeatable tasks while writing scripts * Discover errors earlier and reduce defects * Improve system understanding through visual analysis * Explore alternative earlier in the system lifecycle | * Not targeted towards developers |
| Impact:  BPMN tends to find favor with those coming from a business processing model or business analysis background. Visual dataflow programming languages tend to be favoured by those coming from a software perspective. Tool support also tends to mirror this: bpmn-js(BPMN n.d.) vs Microsoft VPL(Microsoft n.d.).  This research intends to focus on developing a working solution targeted towards developers. | |

Table ‑ BPMN Review

#### Testing

As espoused by Daniel Krutz and Michael Lutz(Krutz& Lutz 2013), testing is an integral and important part of the software development cycle, however it is only a few research papers that touch upon the importance of testing microservices. Indeed most architects and developers still limited by the monolithic approaches of design only think about testing at the application level and highly situational testing at a platform level related to the case study itself. In point of fact due to the ad-hoc of the microservice platform a standardized testing framework is not possible although it has been tried(Heorhiadi et al. 2016; Rahman & Gao 2015). Due to the standardization of the architecture proposed by the platform, a design for a testing framework should be possible, both application testing, and platform level orchestration testing, which is what most developers ignore.

Impact: Developing such a complex testing framework may be a huge undertaking

#### Service Versioning

Application versioning is an integral part of software development. It is the process of assigning unique names or numbers to unique states of the software in development. Within a given version number category (major, minor) these numbers are assigned in increasing order and correspond to new developments in the software.

|  |  |
| --- | --- |
| Advantages Identified | Disadvantages Identified |
| * A complete long term change history of every file * Branching and merging * Traceability: Being able to trace each change to the software and connect it to bug tracking and project management software |  |
| Impact: The versioning of applications is now the norm of the software development lifecycle. The ability to separate earlier versions of the developed project can indeed help developers increase their development speed and time | |

Table ‑ Versioning Review

#### Summary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Research Paper | Virtualization Platforms / Containers | Front end GUI | BPMN | Testing | Handling different service version |
|
| Microservices approach for the internet of things(Butzin et al. 2016) | ✓ | ✓ |  |  | ✓ |
|
|
| Microservices Architecture Based Cloudware Deployment Platform for Service Computing(Guo et al. 2016) | ✓ |  |  |  |  |
|
| Modeling microservices in email-marketing concepts, implementation and experiences(Brüggemann et al. 2014) |  |  | ✓ |  |  |
| Automated Setup of Multi-Cloud Environments for Microservices-Based Applications(Sousa et al. 2016) | ✓ |  |  |  |  |
| Building Critical Applications Using Microservices(Fetzer 2016) | ✓ | ✓ |  |  | ✓ |
|
|
| Open Issues in Scheduling Microservices in the Cloud(Fazio et al. 2016) | ✓ |  |  |  |  |
|
| The Design and Architecture of Microservices(Sill 2016) | ✓ |  |  |  |  |
|
| Challenges in Delivering Software in the Cloud as Microservices(Esposito et al. 2016) | ✓ |  |  |  |  |
|
| Practical Use of Microservices in Moving Workloads to the Cloud(Linthicum 2016) | ✓ |  |  |  |  |
|
| Workload characterization for microservices(Ueda et al. 2016) | ✓ |  |  |  |  |
|
| Gremlin: Systematic Resilience Testing of Microservices (Heorhiadi et al. 2016) |  |  |  | ✓ |  |
|
|
| A microservices architecture for collaborative document editing enhanced with face recognition(Gadea et al. 2016) | ✓ | ✓ |  | ✓ |  |
| Leveraging microservices architecture by using Docker technology(Jaramillo et al. 2016) | ✓ |  |  | ✓ |  |
|
|
| A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development(Rahman & Gao 2015) |  |  |  | ✓ |  |
|

Table ‑ Platform Architecture Research Paper Summary

### Platform Support Architecture

#### Domain Driven Design

Domain Driven Design(Le et al. 2016) proposes having a software mimic a real world process as closely as possible. Close interactions with domain experts accommodate for this. Together the designer and the domain expert build a ubiquitous language (UL): a conceptual description of the system. The point of the this system is for the designer to be able to write down how the process works in such a way that the domain expert is able to verify that it is correct.

|  |  |
| --- | --- |
| Advantages Identified | Disadvantages Identified |
| DDD can help microservices in four major areas:   * **Shared kernel:** meaning two contexts or models share a subset of a domain including code. Plöd notes that this also increases the dependency. * **Customer – Supplier:** with one context acting as a customer asking for features from another context, the supplier. * **Anticorruption layer** (ACL)(Piska 2105): where a client isolates its model from another model using a translation. This is an interesting option especially when migrating to a microservices or a [Self-Contained Systems](https://www.innoq.com/en/links/self-contained-systems-infodeck/) (SCS)(Stranghöner 2015) architecture since the ACL works as a protecting shield against the outside. | * Largely useful for useful when migrating from monolithic to microservice architctures |
| Impact: This process is useful only when analyzing existing processes to come up with a design, not for development | |

Table ‑ Domain Driven Design Review

#### Continuous Delivery

Continuous Delivery(Humble n.d.) is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way.  Teams produce software in short cycles, ensuring that the software can be reliably released at any time. It aims at building, testing, and releasing software faster and more frequently. The approach helps reduce the cost, time, and risk of delivering changes by allowing for more incremental updates to applications in production. A straightforward and repeatable deployment process is important for continuous delivery.

|  |  |
| --- | --- |
| Advantages Identified | Disadvantages Identified |
| * Accelerated time to reach the market * The product can be built the right way * Improved productivity and efficiency * Reliable Releases * Improved Product Quality | * Organizational Challenges * Process Challenges * Technical Challenges |
| Impact: This approach does not directly affect the architecture of the microservice platform in any way, as it focuses on how a developer should develop his system more efficiently, than the platform helping to develop it. However by ensuring that the system is developed in microservice architecture, it’s easily possible to achieve the end result of what CI is aiming for. | |

Table ‑ Continuous Delivery Review

#### Ports and Adapters Pattern

Ports and adapters(Baker 2013), also known as hexagonal architecture, is an attempt to solve the problem of business logic becoming tightly coupled to other dependencies such as client frameworks and persistence. When implemented well, ports and adapters results in little classes with well-defined pieces of functionality. With these pieces of well-defined functionality, the classes are easy to name and the code base becomes comfortable for developers, resulting in reduced cost of maintenance and development. In addition, individual pieces can be mocked out, developed, and tested in isolation. Domain logic can be isolated from side-effect-inducing dependencies, specifically databases and web services.

|  |  |
| --- | --- |
| Advantages Identified | Disadvantages Identified |
| * Describes the architecture of microservices well | * Does not help solve the issue of how the ports are used to communicate with other components |
| Impact: The usage of containers automatically creates a need for communication between each container. While the ports and adapters pattern automatically emulates this, it does not show how to solve it | |

Table ‑ Ports and Adapters Pattern Review

#### Monitoring

Many papers discuss the concept of monitoring, however the context in which they discuses is very different from paper to paper. Certain papers discuss only application level monitoring, while others discuss the monitoring of the microservice itself, still others discuss fault toleration and recovery.

Impact: Creating a design covering so many scenarios may be a herculean undertaking.

#### Summary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Research Paper | Domain Driven Design | Continuous Delivery | Ports and Adapters Pattern | Monitoring |
|
| Microservices approach for the internet of things(Butzin et al. 2016) | ✓ | ✓ | ✓ | ✓ |
|
|
| Microservices Architecture Based Cloudware Deployment Platform for Service Computing(Guo et al. 2016) |  |  |  | ✓ |
|
| Building Critical Applications Using Microservices(Fetzer 2016) |  |  |  | ✓ |
|
|
| Open Issues in Scheduling Microservices in the Cloud(Fazio et al. 2016) |  |  |  | ✓ |
|
| Leveraging microservices architecture by using Docker technology(Jaramillo et al. 2016) |  |  |  | In the form of failure isolation and observability |
|
| Distributed Systems of Microservices Using Docker and Serfnode(Stubbs et al. 2015) |  |  |  | ✓ |
|

Table ‑ Platform Support Architecture Research Paper Summary

### Communication

#### Orchestration

Orchestration is the automated arrangement, coordination and management of computer systems, middleware and services. Its usage is often discussed in the context of SOA, virtualization, provisioning and dynamic datacenter topics. A somewhat different usage and one more closely related to this topic relates to the process of coordinating an exchange of information through web service interactions. Applications that decouple the orchestration layer from the service layer are sometimes called Agile applications(Erl 2005).

Impact: Orchestration in an innate requirement to be met when building microservices. However since all microservices by default need to expose their service in some way, this requirement is partially met all on its own. The rest is taken care of by the service discovery.

#### Choreography

Choreography is a form of service composition in which the interaction protocol between several partner services is defined from a global perspective. Service choreography and orchestration go hand in hand as in orchestration the interaction protocol in at the service level giving a finer level of control between communications in each service.

Impact: The finer level of detail provided by orchestration provides a better ability to control communications than choreography does. Defining a single communication protocol early on in a design limits and forces developers to work around and fit the design to a single protocol during edge cases.

#### Service Discovery

Microservices typically need to call one another. In a monolithic application services invoke one another through language-level method or procedure calls. In a traditional distributed system deployment, services run at fixed, well known locations (hosts and ports) and so can easily call one another using HTTP/REST or some RPC mechanism. However, a modern [microservice-based](http://microservices.io/patterns/microservices.html) application typically runs in virtualized or containerized environments where the number of instances of a service and their locations changes dynamically.

The solution to this problem is to implement a service registry(Peyrott n.d.). The service registry is a database populated with information on how to dispatch requests to microservice instances.

Impact: The implementation of a service registry is a quick an easy solution to a deeply ingrained problem of hosting microservices.

#### API Standardizing(Data modeling and Address Modelling)

As a sub problem of communicating with containers, there is a need to know what containers need to talk about, and what information to pass through. This situation requires APIs to be designed to work either in small subsets of the application arena in which either the API is stable, or to be built to a common self-describing or standardized pattern.

Examples of effective API standards are the RESTful API Markup Language (RAML) (RAML Workgroup n.d.)and Swagger(SmartBear Software n.d.), which has evolved into the Open API Initiative(Linux Foundation n.d.).

###### RAML

RAML is a YAML (Evans n.d.) based language for describing REST APIs. It provides all the information necessary to describe RESTful or practically-RESTful APIs. Although designed with RESTful APIs in mind, RAML is capable of describing APIs that do not obey all constraints of REST (hence the description "practically-RESTful"). It encourages reuse, enables discovery and pattern-sharing, and aims for merit-based emergence of best practices.

###### Open API Specification

The OpenAPI Specification (originally known as the Swagger Specification) is a specification for machine-readable interface files for describing, producing, consuming, and visualizing RESTful Web services.A variety of tools can generate code, documentation and test cases given an interface file. Development of the OpenAPI Specification (OAS) is overseen by the Open API Initiative, an open source collaborative project of the Linux Foundation.

Impact: RAML and Swagger are cross compatible however Swagger has a JSON parser and so is a better fit for this prototype.

## Hardware Architecture

This research intends to focus only on the software aspects of the microservice platform and so therefore even though listed in several papers as important, this research will be pushed to a later date.

## Products

### Docker

Docker(Docker Inc n.d.), is a containerization platform, used to build, deploy and run projects anywhere.

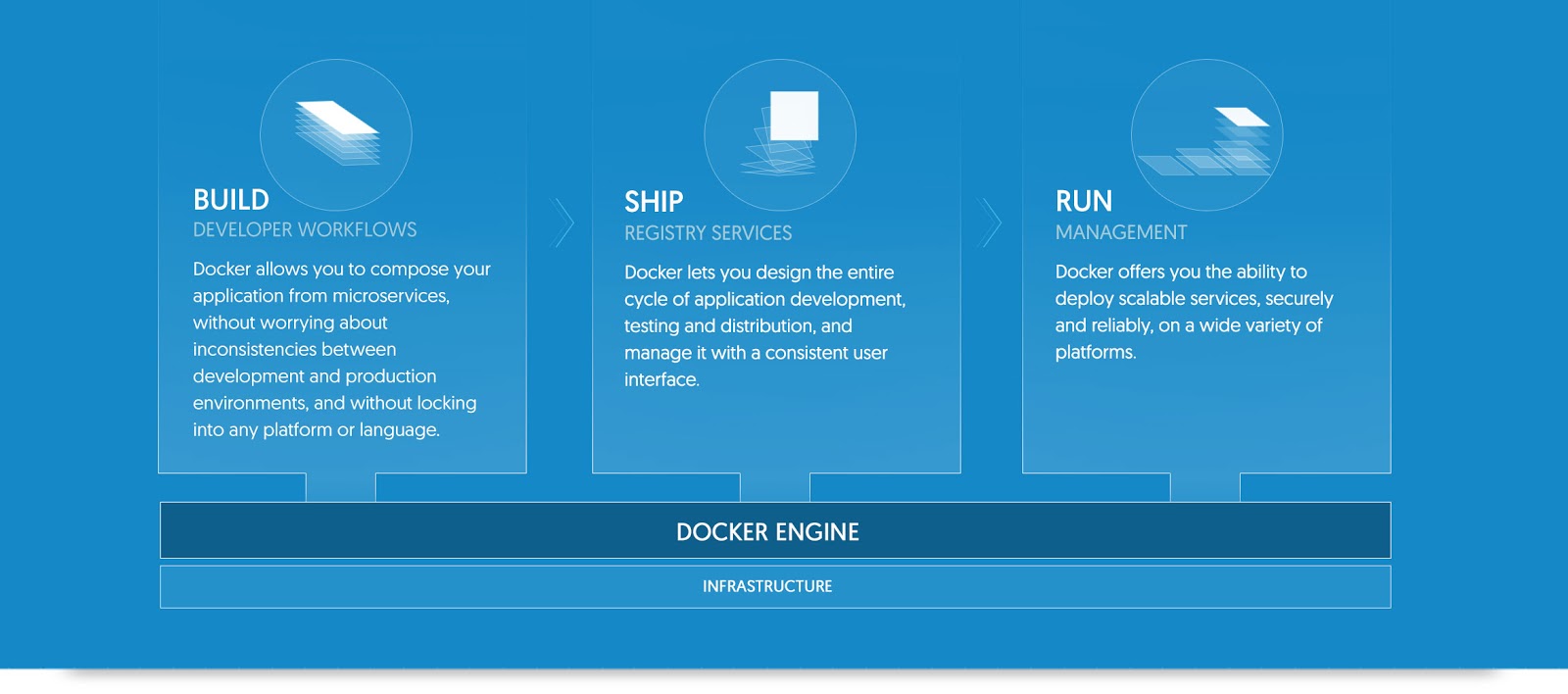


Figure ‑ Docker Capabilities(Docker Inc n.d.)

|  |  |
| --- | --- |
| Advantages Identified | Disadvantages Identified |
| Docker’s main useful feature that makes it relatable to microservices is its Container Engine(Docker Inc n.d.).  Most microservice implementations and case studies recommend the use of containers to separate parts of the whole application(Fetzer 2016; Amaral et al. 2015), and indeed almost every case study breaks the enterprise application down to individual sections/containers (in principle at least if not implementation).  Docker’s second advantage is its API(Docker Inc n.d.), which allows for the remote deployment of containers and applications into containers.  Overall these features along with:   * the fact that Docker is not bound to any VM(Virtual Machine)(Whatis n.d.) allowing it to run almost anywhere * the ability to manage and load balance OS resources for the containers it runs * a clustering feature allowing clusters of managed Docker containers to run over several VMs | * Containers are completely isolated from each other |
| Impact: Even though Docker’s containers can’t talk to each other (this is merely an architectural problem which has already been discussed and ready to be solved), its container technology is a powerful enough concept to be used(since a necessary component of microservices is a container, this has already been discussed before), making it the ideal solution to act as an underlying service management layer for the solution being built over the other presented products which are discussed below. | |

#### WSO2 Microservices framework for Java

This framework(WSO2 n.d.) is an open source product provided by the company WSO2(WSO2 n.d.).It runs on the Spring native programming model (Spring n.d.) and boasts many features including:

Lightweight and fast runtime

* 9MB pack size
* Starts within 300ms
* Designed to be run in container-based environments, such as Docker
* Less than 25MB memory consumption for the WSO2 microservices framework

|  |  |
| --- | --- |
| Advantages Identified | Disadvantages Identified |
| * High performance speed * Scalability * Reliability * Integrated Secuirty | * Requires production support |
| Impact: Its dependence and reliance of not only on other WSO2 products for features(which are irrelevant to the scope of the project) and to function at peak efficiency; but also requires production support from WSO2 (which requires buying a server license) to curate and custode the server and products to run at optimum even though the product is distributed as open source.  This heavy need for reliance on a 3rd party as well as needing to setup irrelevant WSO2 products in a limited time frame makes it unsuitable for use in this project. | |

#### Microservices on AWS

AWS’s (Amazon Inc n.d.) ECS (Amazon Inc n.d.) instances could be used to act like containers of the type much like Docker’s. AWS also provides various admin panels and other AWS products like Beanstalk (Amazon Inc n.d.) and Lambda (Villamizar et al. 2016; Amazon Inc n.d.) and S3 (Amazon Inc n.d.) provide powerful supporting features.

Its database related products in particular make it a much sought after due to the fact that Docker itself does not contain any database related features whatsoever. This lack is usually mitigated by the developer either providing a remotely hosted database, or since Docker can run anywhere a more powerful solution is to combine AWS and Docker.

A typical solution (Amazon Inc n.d.) would be to install Docker in a cluster of EC2 instances, and deploy applications into the Docker containers. This combination allows a variety of advantages:

* Docker contained applications have access to the entire suite of AWS products including the database suite.
* Docker containers can be clustered inside each individual EC2 instance as well, providing all of Docker’s advantages
* The useage of EC2 instances to wrap the Docker containers provides all the benefits of EC2(load balancing, etc) as well as the benefits of the rest of the AWS suite

Impact: An added layer complexity given by combining Docker and AWS is unneeded at the moment, and running microservices on top of Docker provides superior benefits (Docker Remote API (Docker Inc n.d.)) as opposed to running on AWS.

#### Microservices on Google App Engine

Google App Engine(Google Inc n.d.) allows for the building of microservices and deployment to containers much in the fashion of Docker and AWS. A typical architecture of a microservice on Google App Engine would look like below

Google’s advantage is its offering of its entire product suite exposed as APIs for consumption for any applications hosted on its cloud.

Docker allows the projects it manages to deploy to a google VM should a developer choose to do so by using the docker gcloud plugin(Google Inc n.d.).

Impact: Google App Engine and AWS’s main weaknesses are its expensiveness, while Docker can be installed for free on any environment, thus Google App Engine and AWS will be dropped in favour of Docker as the container management platform for the prototype being built.

## Dataflow Programming

By implementing a visual dataflow programming layer this research aims to solve several drawbacks of microservice programming discussed in the introduction. However there exists a myriad form of dataflow programming languages. While they all follow the basic concepts of dataflow programming, their implementations are different enough to warrant a careful review of at least a few of them.

### Products

#### Microsoft Visual Programming Language

MVPL(Microsoft n.d.) is a programming language provided by Microsoft for the Microsoft Robotics Studio. It’s distinguished by other programming languages in the .Net family by being the only true visual programming language.

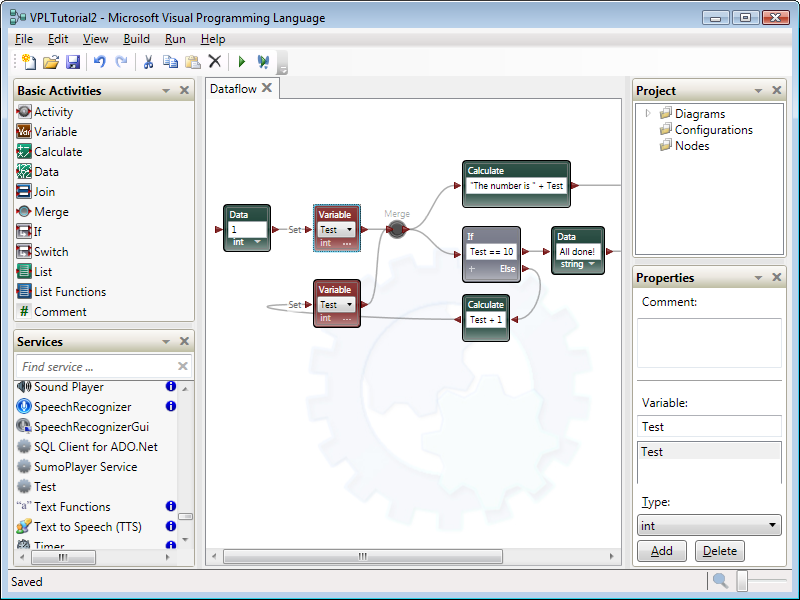


Figure ‑ Tool used for developing Microsoft Visual Programming Programs(Microsoft n.d.)

Mainly targeted at novices, it’s also used by advanced programmers for rapid prototyping.

#### Nextflow

With the rise of big data, techniques to analyze and run experiments on large datasets are increasingly necessary. Nextflow framework is based on the dataflow programming model, which greatly simplifies writing parallel and distributed pipelines without adding unnecessary complexity and letting you concentrate on the flow of data, i.e. the functional logic of the application/algorithm.

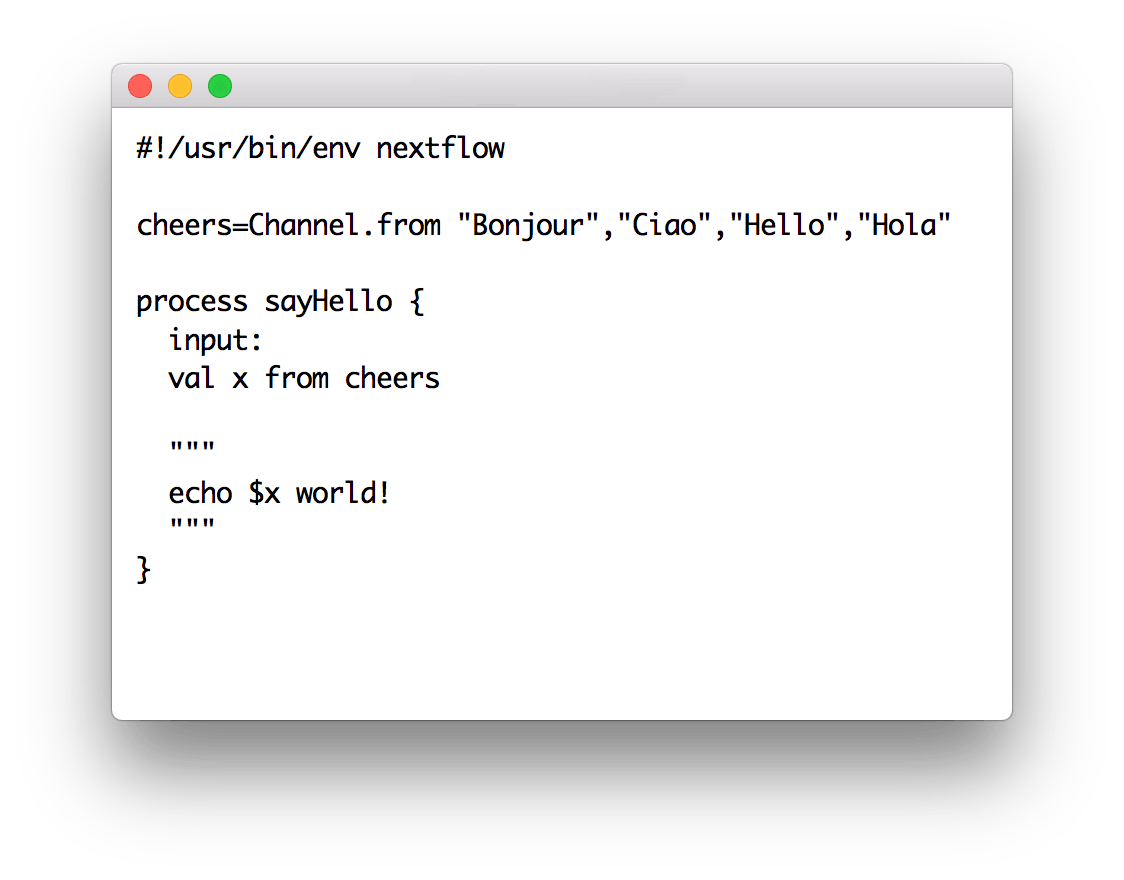


Figure ‑ Developing in Nextflow(Nextflow n.d.)

#### Openwire Library

OpenWire is an open source dataflow programming library that extends the functionality of Embarcadero Delphi and C++ Builder by providing pin type component properties. The properties can be connected to each other. The connections can be used to deliver data or state information between the pins.

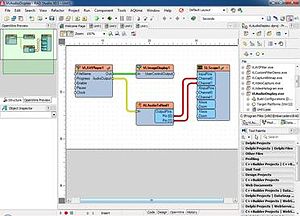


Figure ‑ Developing in Openwire(Openwire n.d.)

Orange is a free software machine learning and data mining package (written in Python). It has a visual programming front-end for explorative data analysis and visualization, and can also be used as a Python library. The program is maintained and developed by the Bioinformatics Laboratory of the Faculty of Computer and Information Science at University of Ljubljana.

#### Review

While all of the products have their strengths, the reason the products above can’t be used is because they run on their own runtime environments. None of the products discussed are extensible enough to be integrated into the microservice architecture being proposed. Therefore the conclusion is to build a visual dataflow language from scratch that can be integrated into the architecture. While the language will be based on general dataflow concepts like ‘State’ and ‘Representation’, its functional components will be implemented based on general programming concepts like ‘loops’, ‘if-else conditions’ and ‘case conditions’. Additionally a few library functions like “REST Calls” will be implemented to make the language easier to use. As a final addition using the proposed ‘Service Registry’, microservices registered in the registry will be used as components themselves to make dataflows. This will in turn help to solve the service orchestration/ choreography/ discovery problem for developers without them having to put in extra development and allowing the microservice platform itself to take care of it.



# Chapter Summary

This chapter mainly focused on finding the best approaches, concepts and techniques to provide a solution to the problem domain. The chapter started by reviewing existing case studies and implementations of microservices solutions. There it was identified that most solutions have many architectural components in common. These components were generally categorized and each architectural principle reviewed both in and out of context to microservices. The best architectural components were selected to be implemented in the proposed microservice framework. Then the chapter analyses existing microservice products, analyzing and reviewing their weaknesses and strengths. The review moves on to review existing dataflow programming products, rejecting all of them due to their inability to run in different runtime environments. The review concluded by choosing to build visual dataflow language modeling general programming components along with a few added custom components. The proposed dataflow component along with the microservice architecture comprises the two major components of the proposed prototype.

# Project Management

Contents

* Chapter Overview
* Project Management Methodology
* Time Allocation
* Constraints and Dependencies
* Potential Risks and Mitigation Plan
* Development Methodology
* Research Methodology
* Chapter Summary

# Chapter Overview

This chapter contains information related to the project management process of the project. It will start from identifying a suitable project management methodology and moving on to time and resource allocations of the project. Then it will discuss about the possible risks along with appropriate mitigation plans. Finally it will lead to a discussion about selecting a suitable software development methodology for the project.

# Project Management Methodology

Any project would have scope, time and cost as the constraints of the project and it is important to manage the above mentioned constraints properly to produce a quality output from the project undertaken. In a research project, managing those constraints going to be harder compared to an industrial project due to frequent requirement fluctuations, which highlights the importance of a suitable project management methodology. PRINCE2 is a widely used project management methodology which encompasses the high level management, control and organisation of a project. It was decided to adopt PRINCE2 as the project management methodology due to the previous exposure and experience with it.

# Time allocation

Following table represents an overview of the identified main tasks of the (Refer Appendix C for Gantt Chart) project. Research projects require the literature review process to be carried till the end of testing phase due to frequent requirement fluctuations of research projects. Subsequently the documentation of project will too spread throughout the project life time as it will be the ultimate deliverable of this exercise

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Name | Duration | Start | Finish | Predecessors |
| 1 |  | **55d** | **09/15/2016** | **11/30/2016** |  |
| 2 | Research Domain | 49d | 09/15/2016 | 11/22/2016 |  |
| 3 | Submit TOR | 6d | 11/23/2016 | 11/30/2016 | 2 |
| 4 | **Literature Review** | **33d** | **12/01/2016** | **01/16/2017** |  |
| 5 | Research usecases of microservices | 22d | 12/01/2016 | 12/30/2016 | 3 |
| 6 | Research implementation methods of microservices | 22d | 12/01/2016 | 12/30/2016 | 3 |
| 7 | Research dataflow programming paradigms | 22d | 12/01/2016 | 12/30/2016 | 3 |
| 8 | Evaluate current microservice products | 22d | 12/01/2016 | 12/30/2016 | 3 |
| 9 | Create and submit Literature Review | 11d | 01/02/2017 | 01/16/2017 | 5,6,7,8 |
| 10 | **Requirements Gathering** | **31d** | **01/17/2017** | **02/28/2017** |  |
| 11 | Gather Requirements | 24d | 01/17/2017 | 02/17/2017 | 9 |
| 12 | Prepare and submit SRS | 7d | 02/20/2017 | 02/28/2017 | 11 |
| 13 | **Design** | **23d** | **03/01/2017** | **03/31/2017** |  |
| 14 | Create Design for Microservice Platform | 18d | 03/01/2017 | 03/24/2017 | 12 |
| 15 | Create Design for Dataflow Framework | 1d | 03/01/2017 | 03/01/2017 | 12 |
| 16 | Prepare and submit Design report | 5d | 03/27/2017 | 03/31/2017 | 14 |
| 17 | **Development** | **44d** | **04/03/2017** | **06/01/2017** |  |
| 18 | Develop Prototype | 40d | 04/03/2017 | 05/26/2017 | 16 |
| 19 | Prepare and submit Implementation Report | 4d | 05/29/2017 | 06/01/2017 | 18 |
| 20 | **Testing** | **21d** | **06/02/2017** | **06/30/2017** |  |
| 21 | Test Prototype | 17d | 06/02/2017 | 06/26/2017 | 19 |
| 22 | Prepare and submit Testing Report | 1d | 06/30/2017 | 06/30/2017 | 21 |
| 23 | Evaluation | 13d | 07/03/2017 | 07/19/2017 | 22 |
| 24 | Prepare and submit final Report | 8d | 07/20/2017 | 07/31/2017 | 23 |

Table ‑ Gantt Chart

# Constraints and Dependencies

The successful completion of project will be dependent upon the following constraints and dependencies.

* **Time constraint** - considered as a major challenge for the successful completion of the project as the whole software development life cycle has to be carried individually.
* **Lack of prior knowledge** – due to lack of knowledge on microservices and dataflow programming the development of the proposed system may be hindered.

# Potential Risks and Mitigation Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Risk Id 1 | Failure to keep with the up-to-date domain knowledge & technology changes | | |
| Risk level | High | Occurrence Frequency | High |
| Description | Microservices is a broad research domain where new ideas and concepts are brought forward on a regular basis making it difficult to keep up with the most up to date knowledge of the domain. It has to be accepted that the surrounding technologies too may change over the course of the project time schedule .This will present a risk that the system is not be developed using the most up to date domain knowledge nor using the latest technologies | | |
| Mitigation | * Check for any new work done on the domain area on a weekly basis * Check for technology changes on a weekly basis. * Keep frequent communication with domain experts about the latest developments. | | |
| Risk Id 2 | Constant and repeated changes to requirements | | |
| Risk level | High | Occurrence Frequency | High |
| Description | As the nature of a research project there would be constant changes to the requirements in every phase of the software life cycle. This is considered as a risk project as it may be difficult to manage all the requirement changes within the allocated time period. | | |
| Mitigation | * Prepare ahead for the changes * Prioritize the changes and try to tackle only the critical changes | | |
| Risk Id 3 | Data lost due to hardware, software and system failures | | |
| Risk level | High | Occurrence Frequency | Low |
| Description | During the course of the project the data of the project can get lost due to hardware, software and system failures and it may prove to be a critical threat to the successful completion of the project | | |
| Mitigation | * Maintain daily backups * Carryout continuous testing and debugging of the software. | | |

# Development Methodology

In order to avoid schedule and cost overruns and to mitigate frequent requirement changes of the project a suitable development methodology has to be adopted. With the aim identifying a suitable development methodology, key characteristics of several software development methodologies are critically evaluated. Following table shows the characteristics of several frequently used software development methodologies and the below evaluation is based on the available resources, nature of the project and ability to meet the identified requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Development Methodology | | | | | |
| Characteristics | Waterfall  Methodology(Bassil 2012) | Rapid Application Development(Martin & James 1991) | Spiral Methodology(Boehm 1988) | Agile(Taylor et al. 2009) | Prototype Development Methodology(Alavi 1984) |
| Top down development method with rigid phases | Iterative Development | Iterative Development | Iterative and incremental Development | Prototype is developed based on currently known requirements |
| Linear Module | Software Prototyping | Continuous Refinement of final product | Agility for rapid changing | Higher rate on customer interaction |
| All requirements should be previously known | Fast development with minimum planning and high quality systems | Emphasises Risk Analysis | Accelerated Delivery and Customer Interaction |  |

Table ‑ Development Methodologies

Waterfall method is suitable for projects with a stable set of requirements but in a research project by nature requirements change during each phase of the software development life cycle. Since linear model with rigid phases doesn’t support requirement fluctuations of experimental and research development, traditional waterfall approach is not suitable for a research project. Therefore, this methodology will be discarded in favor of a methodology where a more iterative approach is applied.

When considering the Rapid Application Development methodology, it can be seen that it uses minimal planning and then quickly moves in to prototype development at an early stage of the project duration. It is evident that it is a must to have a certain level of understanding of the final products’ functionalities to start developing the prototype at an early stage. But in a research project functionalities of the prototype can’t be identified with a higher degree of certainty at the start of the project and it can be concluded that this methodology is not suitable for a research project due to the constraint of not knowing the functionalities of the prototype in an early of the project.

Agile methodology uses iterative and incremental development strategy which includes lot of testing, customer interactions and discussions throughout the development phase which may result in time overrun. Additionally Agile methodology is more suited to handle projects with projects with massive scopes and larger number of stakeholders on various levels of an organization. Since the project undertaken has a major constraint on the time availability and neither involved with a huge scope nor a larger number of stakeholders it can be concluded that Agile methodology isn’t suitable to handle the project in hand.

Similar to the Rapid Application Development methodology Prototype methodology too focus on moving into the development of the prototype at an early stage of the project and poses the same problem like in the Rapid Application development methodology as the functionalities of the prototype can’t be determined at an early stage of a research project. Additionally it may increase the complexity of the system and may lead to scope enhancements of the system due to the higher rate of customer interaction leading to time overruns. Therefore it can be deduced that Rapid Application Development methodology isn’t suitable to conduct the project undertaken.

The Spiral methodology supports continuation of cycles of the software development life cycle without clear termination conditions which would help to counter frequent requirement fluctuations which would lead to timely delivery of the prototype. Additionally the Spiral methodology helps to analyse the risks that would affect the final product and finding those risks in advance can lead to speed up the development process. As the major challenges of the project in hand are the time constraint and frequent requirement fluctuations it can concluded that Spiral methodology would be the most suitable development methodology.

# Research Methodology

As highlighted by Dubois and & Gadde(Dubois & Gadde 2002) a research can be categorized in to two categories known as inductive researching and deductive researching. Deductive approach is aimed at proving and testing a hypothesis while the inductive approach is concerned with generation of new theory emerging from data. The project in hand falls in to the deductive researching approach as the aim of the project is to prove and test that a software agent based recommender system can solve the limitations mentioned in the introduction chapter.

# Chapter Summary

This chapter initially highlighted the importance of a suitable project management methodology to get best result out of the project and PRINCE2 which is a widely used project management methodology was chosen as the most suitable project management methodology due to the previous experience and exposure with it. Next the entire project was broken down into different task and proper time allocation was done, so that work can be done accordingly and parallel. Then followed a descriptions of identified constraints of the research and possible risks that can occur during the course of the project were identified and appropriate mitigation plans were discussed. After that to identify a suitable development methodology for the project several development methodologies were evaluated and it was decide to adopt the Spiral development methodology due to its flexibility to support frequent requirement changes that can occur during the project life cycle. Finally this chapter discussed choosing a research methodology that needed to be adopted for the successful completion of the project and deductive research methodology was adopted over inductive methodology due to the reason that this project was about creating a new microservice platform to alleviate developer overhead and concerns.

# System Requirements Specification

Contents

* Chapter Overview
* Requirements Elicitation Process
* End User Questionnaire
* Formal Interviews
* Findings
* Stakeholders
* Context Diagram
* Use Case Diagram
* Functional Requirements
* Non Functional Requirements
* Chapter Summary

# Chapter Overview

The previous chapter presented a detailed review of the problem domain & existing literature in relation to microservices and dataflow programming. This chapter starts by presenting the process untaken to gather requirements from identified stakeholders for the proposed platform. Next this chapter will divulge deeper into several requirement analysis models carried out to analyze the raw data gathered and identify requirements of the system. The results from the requirements gathering techniques are analyzed to construct a set of functional and non-functional requirements. The requirements are used to construct a set of use case diagrams and subsequent use cases that examine key aspects of the project. The chapter concludes with the domain models which reflect the functionality identified in the use cases.

# Requirements Elicitation Process

In order to ensure efficiency of the requirement gathering process and mitigate the limitations of each single requirements elicitation method, several approaches were utilized parallel to gather requirements from various stakeholders. The below table provides an overview of the factors that lead to the selection of each utilized requirement elicitation method.

|  |  |
| --- | --- |
| Method No 1 | Literature Review |
| The literature review carried out on the current status of microservice platforms and dataflow programming environments to analyze the current progress and limitations of both systems. | |
| Advantages | * Facilitate to identify and better understand the current limits of both platforms. * Paved way to identifying certain areas that have be included in the questionnaire and interview process for better clarifications |
| Disadvantages | * Due to delays in publications literature material may not present the latest developments and limitations of the related domains * Time consuming effort due to the time and effort required review the vast amount of literature available |
| Method No 2 | Questionnaire |
| A questionnaire focusing on identifying the end user requirements for a microservice platform was prepared. Since the primary users of this platform would be developers, copies of the questionnaire was made available online and placed on sites where developers liked to frequent often. | |
| Advantages | * Comparatively time saving method to other elicitation methods * Ability to cover geographically, communality dispersed users * Ease of tabulation and comparison of the feedback due to the standardization of questions |
| Disadvantages | * Success of identifying the correct requirements depends on the honesty of the participants * May encumber participants from sharing additional information due to standardization of the questions. * Difficulties faced when comprehending answers given to open ended questions |
| Method No 3 | Formal interviews |
| A series of formal interviews were carried out with the domain experts of the microservice platforms with the aim of identifying current processes adapted to when developing a microservice | |
| Advantages | * Ease of clarifying doubts with the experts while the interview is going on * Enabling the elicitation of measurable requirements and guideline for the implementation |
| Disadvantages | * Inability to reach a wider audience of domain experts due to time taken to conduct an interview * Subjective interpretation of the problem by the experts |
| Method No 4 | Self-evaluation |
| Several self development sessions of microservices were carried out in a controlled environment in order to experience and observe limitations of microservices | |
| Advantages | * Direct insight into the practical limitations faced by developers * Ability to identify new limitations that may not have been identified or ignored by the research experts. * Can be used to validate the identified requirements |
| Disadvantages | * Certain requirements may be ignored be subjective to self-experiences * Developer inexperience may hinder the evaluation process with bias |

Table ‑ Summary of Requirements Gathering techniques

Observations as a requirements gathering process was rejected due to it being a too time consuming process to use. Development of a project could take anywhere between 3 months to several years as well as containing main team members and several moving parts. Therefore observing a full project development lifecycle may be too consuming of a process.

## End user questionnaire

### Structure of the questionnaire

The questionnaire distributed among the potential end users of the system (the developers) contained questions to identify:

* User Behavior when developing microservices
* Pain points during development of microservices

### Limitations of the Questionnaire Process

|  |  |
| --- | --- |
| Limitation Id 1 | Limited response |
| Though the questionnaire was made available to a larger number of potential end users of varying development methodologies not all of the questionnaires were returned back and a substantial amount of the returned were not adequately completed. Therefore the gathered responses may not represent the absolute view of all the end users. | |
| Limitation Id 2 | Feedback credibility |
| Since the questionnaires were filled by individuals at their own free will there is no way to authenticate the credibility of the feedbacks given. Therefore the gathered responses may not represent the absolute view faced by the developers. | |

Table ‑ Limitations of Questionairres

### Requirement elicitation of questionnaire

Equation ‑ Response Rate of Questionnaire

According to the above figure the successful response rate was above 81% for the normal user questionnaire and with the respondent quantity and percentage it can be concluded that the questionnaire process was successful.

#### How do you host your microservice?

This question was asked to gather the ways developers use to host their developed microservices. This requirement was idenfied during the product review of the Literature Research chapter. The below chart shows the results:

Figure ‑ Preferred Hosting Solutions

As the chart shows: an overwhelming majority of developers rely on Docker for hosting their microservices. This is consistent with earlier research on Docker being the goto system for hosting microservices. This response directly helped identify (FR2 and FR3). The functional requirements are described in greater detail below

#### How do you deploy your microservice?

While the previous question asked about the hosting location, this question asks about the hosting location, this question asks the hosting method. This question is related to both the development language and the hosting method and was indentified during the product review of the Literature Review Chapter.

Figure ‑ Preferred Deployment Solutions

The chart shows the overwhelming majority is again Docker. This is the expected response as Docker is both a hosting solution as well as a location. The next highest are tomcat and IIS, Tomcat and Apache for java developers, and IIS for .NET developers, the rest use Kubernetes which is language agnostic like Docker. This response directly helped identify (FR2 and FR3).

#### What Language do you use to develop your microservice?

This question was asked to gauge the variance of languages used to develop microservices in.

Figure ‑ Preferred Languages

As can be seen in the charts above there is no one specific overwhelmingly used language for microservice development. The result of this is that whatever deployment solution being used has to be language agnostic as well. Docker performs this function nicely which is why as seen in the previous responses Docker is overwhelmingly used as the container solution. This question was used to identify FR1.

#### What IDE do you use for development?

This question was asked to gauge which IDEs were used by developers to develop microservices in. All the popular IDEs were shown in the question. This question is relevant due to the fact that this research aims to build a visual dataflow model of developers to design, build and develop the microservice in.

Figure ‑ Preferred IDEs

The popular traditional IDEs are the ones mainly used for development, with other methods of development neglected. This is due to developers mainly used to using programming languages to develop in.

This question was used to identify FR1. This question was also used to validate the feasibility of whether or not developing using a dataflow language was a viable alternate solution to the current traditional development. The use of such a large and varied list of IDEs seems to portray that developers don’t mind adding another solution in which to develop in, given that a largely proportionate number of responses included multiple IDEs.

#### Do you use a service registry to handle inter-service communication, if not please specify your method

This question was asked as part of the section on communication: orchestration, choreography, address modeling, and data modeling.

Figure ‑ Preferred communication method

Very few developers spend time developing service registry solutions. Most developers in the interest of saving time, prefer to hard code the API location in some form, predominately in the form of configuration files. However since the platform is automated, hard coded locations are not a suitable solution for the proposed platform.

This question directly impacted the identification of FR6, FR7, FR8, FR9, FR10, and FR15.

#### Explaining the service versioning strategy

This question was presented in the form of a short answer. The overwhelming majority responded as using semantic versioning(Preston-Werner 2013) in their strategies or a variation of it.

This question directly impacted FR5.

#### What level of testing do you do?

This question was asked with the intention of developing a testing framework for the proposed platform. This question directly impacted FR4.

Figure ‑ Preferred Testing Solutions

#### If so do you use a testing framework?

This question is merely an extension of the previous one, as testing frameworks are predominantly used by developers for unit testing.

Figure ‑ Preferred Testing Method

An almost equal number of developers both use and do not use testing frameworks, possibly due to certain types of testing like scalability and fault tolerance not needing them.

#### If Yes, please specify (comma separated for multiple frameworks)

This question is the continuation of the previous one, asking about the names of the frameworks used. This was presented as an open question, as there are a myriad of testing frameworks for every language. As expected the answers too were myriad, with developers naming multiple frameworks for different languages.

#### Explain your continuous integration strategy

This question was asked with the requirement of continuous integration that was identified from the literature review in mind.

Figure ‑ Preferred CI Strategy

An almost equal number of developers both did and did not use CI, those that did were predominantly divided between Jenkins and Codeship; the two most popular CI solutions.

This quesion directly impacted the identification of FR6.

#### Do you maintain multiple build environments?

This question was asked with the requirement of continuous integration in mind.

Figure ‑ Preferred Environment Solution

Many developers maintain at least two or more environments, for dev and QA.

This question helped identify many requirements namely: FR10, FR11, FR12, FR13, FR14, FR15, FR16, FR17.

#### If so, How do you propogate the microservices around the environment( eg: through CI tool)

This question was asked in a long form. The responses of the developers who did not use a CI tool were almost universally to copy the release by hand and deploy it (or some variation of it).

This question along with the above one helped to identify the above mentioned requirements.

#### Do you run multiple versions of the same service

This and the next question was asked with the intention of finding out how different versions of the same service were run side by side

Figure ‑ Whether or not multiple versions run

The answers were almost overwhelmingly in the positive

This question along with the above one helped to identify the above mentioned requirements.

#### If so please specify how they are hosted side by side

This question was asked in a long form. Developers gave answers in what can be classified as two major categories:

* They were renamed as a different service (usually with the name being with regards to the service’s purpose)
* They were hosted by their version numbers.

This question along with the above one helped to identify the above mentioned requirements.

## Formal Interviews

### Structure of the interview questions

Several formal interviews were conducted with a few developers, both online and in person with the target of identifying the pain points when it comes to developing microservices

### Limitations of the interview process

|  |  |
| --- | --- |
| Limitation Id 1 | Preservation of self-interests |
| Some developers were not enthusiastic about answering certain questions as the questions themselves seemed to put them out of their comfort zone. Many developers had overtime time come up with their own habits and workflows to developing microservices, and sometimes the workflow/habits were not the most optimal way of development. Suggesting even minor improvements in order to better gather requirements seemed to put both developers and managers off. Therefore the gathered responses may not represent the absolute view of the whole development method. | |
| Limitation Id 2 | Lack of knowledge |
| Certain managers were not technically competent enough to understand the functionalities of developing microservices. Therefore they were unable to provide a clear feedback on a number of technical questions. | |

Table ‑ Limitations of using interviews

## Findings derived from requirements elicitation process

The identified summary of requirements are shown in the below table. The expounded requirements list can be found later on under Functional Requirements and Non Functional Requirements.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Finding | Literature Review | Questionnaire | Formal Interview | Self Evaluation |
| 1 | Usage of containers | ✓ | ✓ | ✓ | ✓ |
| 2 | Testing Framework | ✓ | ✓ |  | ✓ |
| 3 | Service Versioning | ✓ | ✓ | ✓ |  |
| 4 | Continuous Delivery | ✓ | ✓ | ✓ |  |
| 5 | Communication Framework | ✓ | ✓ |  | ✓ |
| 6 | Data Modeling | ✓ | ✓ |  | ✓ |
| 7 | Address Modeling | ✓ | ✓ |  | ✓ |

## Stakeholders

### Stakeholders Onion Diagram

onion model.png

Figure ‑ Onion Diagram

Above onion diagram represents the identified stakeholders and their roles in proposed solution.

### Roles

In a typical development environment:

* The project manager has overall responsibility for the entire project and makes the management decisions.
* The architect comes up with the overall design of the project and submits to the tech lead and manager to approve.
* The tech lead decides which technologies to use in the project.
* The developer works directly with the microservice platform to develop the project
* QA and QC assure and mange its quality.
* The end users are the ones who directly use the microservice after its developed and released.

|  |  |  |
| --- | --- | --- |
| **Stakeholder** | **Role** | **Viewpoint** |
| End User | Operator/ Functional Beneficiary | Expects to use the developed microservice  Execute the microservice |
| Developer | Operator | Expects to develop the microservice  Wants to use the microservice platform to develop the microservice |
| QA/QA | Functional Support | Expects to test and verify authenticity and quality of microservice  Wants to use the testing platform to test the microservice |
| Architect | Functional Support | Expects to design the microservice to conform the platofrm’s requirements  Wants to design the microservice using the platform |
| Tech Lead | Functional Support | The tech lead analyses the microservice design, other available and deployed microservices to combine into a new microservice |
| Project Manager | Managerial beneficiary/ Intellectual | Handles the managerial decisions of developing the microservice. |



## Context Diagram

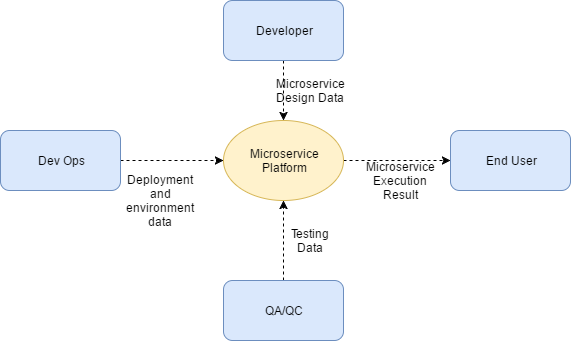


Figure ‑ Context Diagram

## Use Case Diagram



Figure ‑ Use Case Diagram

## Use Cases

|  |  |
| --- | --- |
| **Use case name** | Draw Flow of Microservice using dataflow programming components |
| **Description** | This use case is to show the process where a microservice is drawn using the tools provided by the dataflow layer of the microservice platform |
| **Participating actors** | Developers |
| **Pre-conditions** | 1. System should be functional |
| **Extended use cases** | Test Microservice, Deploy microservice to management platform |
| **Included use cases** | None |
| **Main flow** | 1. Developer drags a component from the toolbar 2. Developer drops the component onto the canvas 3. Developer drags and drops another component to the canvas 4. Developer selects a component 5. Developer drags a connector to another component 6. Developer sets the configuration for the connector 7. Developer saves the design |
| **Alternative flows** | None |
| **Exceptional flows** |  |
| **Post conditions** | System saves and stores the designed microservice layout |

|  |  |
| --- | --- |
| **Use case name** | Test Microservice |
| **Description** | This use case is to show the process where a microservice is tested using the tools provided in the dataflow layer |
| **Participating actors** | Developers,QA & QC |
| **Pre-conditions** | 1. System should be functional 2. There should be a designed microservice |
| **Extended use cases** | None |
| **Included use cases** | None |
| **Main flow** | 1. Developer opens up existing developed microservice project 2. QA opens up the testing panel 3. Developer sets initial testing configuration 4. QA sets test case inputs 5. QC executes the test 6. QA and QC views the test case result |
| **Alternative flows** | None |
| **Exceptional flows** |  |
| **Post conditions** | System saves and stores the test cases |

|  |  |
| --- | --- |
| **Use case name** | Deploy Microservice |
| **Description** | This use case is to show the process where a microservice is deployed to the microservice platform |
| **Participating actors** | Developers, DevOps Engineers |
| **Pre-conditions** | 1. System should be functional 2. There should be a designed microservice 3. The microservice should be tested and pass all test conditions |
| **Extended use cases** | None |
| **Included use cases** | 1. Deploy to staging/QA environment |
| **Main flow** | 1. Developer opens the microservice project 2. Developer checks if tests have passed 3. DevOps presses the deploy button 4. DevOps selects the environments 5. DevOps sets the final configuration values 6. DevOps deploys the project |
| **Alternative flows** | None |
| **Exceptional flows** |  |
| **Post conditions** |  |



## Functional Requirements

**Requirement Prioritization**

Due to time and resource restrictions it might be difficult to implement every identified requirement in the requirement elicitation process. So the identified requirements are prioritized in order to identify most important and less important requirements. Following table shows the priority levels and the descriptions of it.

|  |  |
| --- | --- |
| **Priority level** | **Description** |
| Critical (**C**) | Functional requirements of the ‘Critical’ category represent the core functionality of the system, and it is mandatory to be implemented |
| Important (**I**) | The functional requirements of this category are not essential, but they are considered to be necessary. |
| Desirable (**D**) | The requirements which are intended to implement in further developments or out of the scope for the project. |

The following table shows the identified functional requirements along with the priority levels

|  |  |  |
| --- | --- | --- |
| **FR No** | **Use case Mapping** | **Priority Level** |
| FR1 | Developers should be able to use the dataflow tool to drag and drop components to develop the microservice | C |
| FR2 | Developers should be able to use the microservice platform to deploy the microservice to different environments | C |
| FR3 | The microservice platform should deploy the microservice into a Docker container | C |
| FR4 | QA/QC should be able to use the platform to test the service before deploying | I |
| FR5 | The microservice platform should take care of deploying microservices with adequate service versioning | I |
| FR6 | The microservice platform should take care of Continuous Delivery when deploying the microservice | D |
| FR7 | The microservice platform should posses a service registry to handle communication | C |
| FR8 | The microservice platform should use Swagger to expose the metadata of the input and output of microservice | I |
| FR9 | The service registry should be able to manage multiple environments for each microservice | I |
| FR10 | Dev Ops should be able to enter configuration details of different environments for each microservice | I |
| FR11 | The health of each microservice should be easily monitored | D |
| FR12 | If the health of a microservice is in trouble an alarm should fire | D |
| FR13 | The microservice platform should be able to deploy to multiple platforms simultaneously | D |
| FR14 | The lifecycle of the microservice should be managed by the platform | D |
| FR15 | It should be able for a developer to discover another microservice and use as a component in his microservice using the service registry | C |
| FR16 | The microservice platform should handle the Release management of the microservice | D |
| FR17 | API tagging and taxonomy should be possible with the microservice platform to make it easier to find a microservice in the service registry | D |



## Non Functional Requirements

Following are the identified non-functional requirements of the **s**ystem. The purpose of non-functional requirements is to provide a better user satisfaction, improve performance and to enhance maintainability.

|  |  |
| --- | --- |
| NFR 1 | Usability |
| The system should ensure higher level of usability by making the prototype usable for new users with minimum training. The user interface has to be simple and visually appealing to use. Help messages will be provided where necessary to improve usability. A new user should take maximum of 1 day to get familiar with the functionalities and the capabilities of the proposed system. | |
| NFR2 | Scalability |
| Another major non-functional requirement of the proposed system is to maintain a higher level of scalability. It’s expected that the workload of the system will increase as time goes on with the number of services increasing and the number of API consumers increasing as well. So the system should be scalable enough to handle to increasing workload in a satisfactory manner. | |
| NFR3 | Performance |
| System should respond quickly for the API consumers and work properly without crashing on all types of devices and platforms. | |
| NFR4 | Uptime |
| The system should be available for as long as possible. API consumers that use services will fail should the proposed system go down. Therefore keeping the system up and running for as long as possible without any downtime is mission critical. | |

## Chapter Summary

This chapter mainly focused on gathering correct and adequate requirements expected from the proposed microservice platform system. Chapter started with a detailed description of the various techniques that were used to gather requirements from identified stakeholders. Questionnaires and interviews were the two main elicitation techniques used, other than the literature review, and self-evaluation techniques. Later all the collected results were analyzed and documented to refine user requirements, identify the project scope and formulate a solution to the problem. The key stakeholders and their roles were identified via an onion diagram. Use case and Use case descriptions were included into this chapter to clearly identify the stakeholders’ interactions and expectations from the system. Then the chapter went to document a comprehensive list of functional and non-functional requirements. Next chapter will be the System Architecture and Design, which will be focusing on the detailed system design of the prototype based on the requirements identified from this chapter.

# System Design

Contents

* Chapter Overview
* High Level Design
* System Design
* Domain Model of the Microservice Platform
* Class Diagram
* Sequence Diagram
* Dataflow Tool
* Microservice Platform
* Chapter Summary



# Chapter Overview

Having discussed requirement elicitation process in the previous chapter, this chapter focuses on the architecture and the design of the Microservice system. Chapter will first discuss the high level architecture and workflow of the system, introducing the sub components which form the complete system. This will be followed by the component level design of the system which focuses the functionalities of each sub component in order to meet the requirements in requirement specification. Finally, integration process of these sub components into main system will be discussed.



# High Level Design

The high-level design provides a fairly accurate picture of the end goal and the process that is needed to get there, allowing filling in the details later on the exact steps and configurations that will be needed for everything to fit together.



## Rich Picture of the Microservice Platform

A Rich Picture is an approach to explore a situation and define a high-level solution and then express the solution through diagrams to create a preliminary design model that can be further expanded with an appropriate design methodology. The below figure illustrates the Rich Picture diagram of the microservice platform.

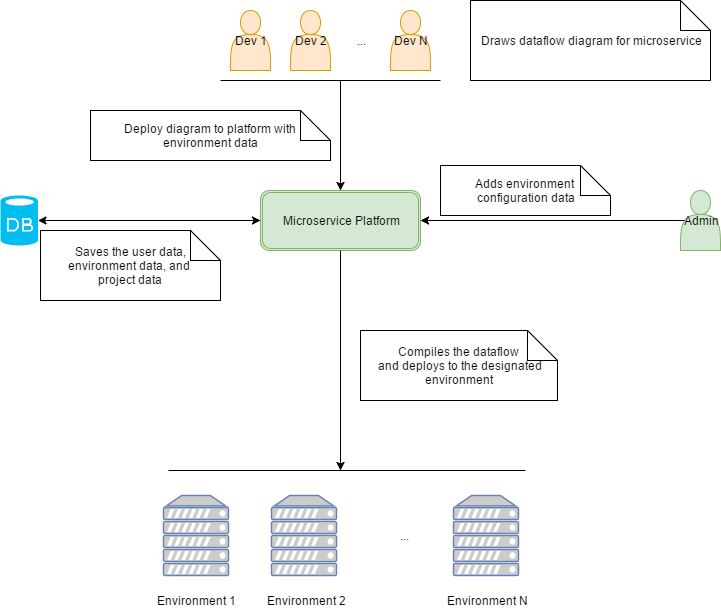


Figure ‑ Rich Picture

According to figure 4.2.1 the main stakeholders are the developers. The developers create a microservice project and build the dataflow diagram of the microservice. They then deploy the microservice to the platform along with the environment location they want the microservice deployed to. The admin is a secondary stakeholder that manages the configuration and environment data for the microservice platform. The platform receives the dataflow, compiles it into a microservice and deploys it to the environment.

## High Level Architecture

Three - Tier architecture is an industry-proved software architecture model which support modeling of enterprise-level client/server applications. Main advantage of the three-tier architecture is the independency of the three tiers from allowing any of the three tiers to be upgraded or replaced independently. According to (Marston n.d.) three-tier architecture is capable of resolving issues like scalability, security, fault tolerance and due to the above mentioned reasons the high level architecture of the proposed Microservice System was modeled using the three-tier architecture.

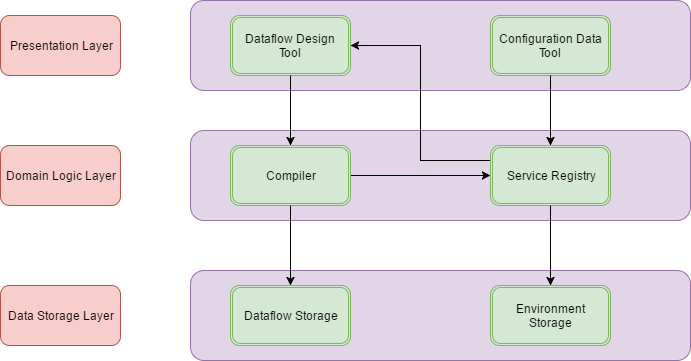


Figure ‑ High Level Architecture Diagram

The presentation layer exposes two modules:

**Dataflow Design Tool:** This tool is what developers will use to build the microservice and deploy it.

**Configuration Data Tool:** Developers and Dev Ops engineers add configuration data to the platform which gets added to the service registry component.

The domain logic layer houses two major components:

**Compiler:** The compiler takes in the deployed dataflow and converts it into a microservice. The dataflow itself is passed onto the dataflow storage service. The microservice itself is deployed onto the environment and the service registry is updated with relevant details.

**Service Registry:** The service registry records microservices deployed to environments and pass them along to the dataflow design tool. This allows developers to reuse microservices inside of other microservices.

The data storage layer houses two components:

**Dataflow Storage:** This component stores the developed dataflow for later reuse and or modification.

**Environment Storage:** The environment configurations are stored here to be used in projects

# System Design

## Selection of Design Methodology

Design methodology is a set of procedure that one follows from the beginning to the completion of the software development process. The nature of the methodology is dependent on several main factors like software development environment, type of the software being developed, the requirements of the users, and the time schedule. Though several design methodologies are available the most widely used two methodologies by both academia and industry are.

* Structured Systems Analysis and Design Method (SSADM)
* Object-Oriented Analysis and Design Method (OOADM)

**Application of structured systems analysis and design methodology**

SSADM approach supports development of function oriented software. Though the SSADM is not widely used in the industry as key design methodology due its incapability in modelling complex and larger software it is a respectable analysis and design methodology when it comes to modeling fairly small and simple software.

**Application of object oriented analysis and design methodology**

OOADM approach supports development of object oriented software. OOADM approach is widely used in the industry as key development methodology due its capability in modelling complex and larger software. OOADM is capable of breaking the complex software system down into its various objects, combining the data and the functions that operate on the data into a single unit.

Since the implementation of the platform depends on JavaScript and NodeJS (refer to the Implementation chapter) it was decided to model the rest of the system using the OOADM approach and Unified Modeling Language (UML) was selected as the technique for representing the model.



# Domain Model of Microservice Platform

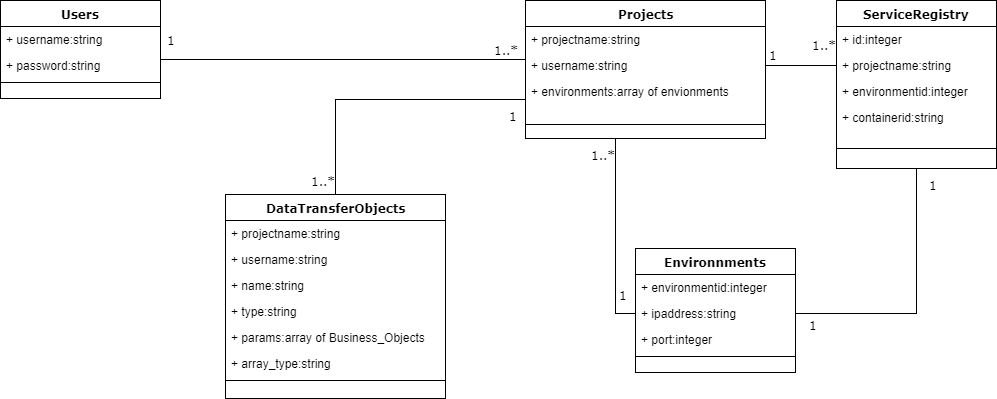


Figure ‑ Domain Model of Microservice Platform

|  |  |
| --- | --- |
| **Class** | Description |
| **Users** | All users (developers, dev ops engineers, etc) are modelled through this class |
| **Projects** | Developers create projects. These projects house the dataflow diagram, and deployment environments. |
| **ServiceRegistry** | All deployed microservices get added to the service registry, to allow developers to add these microservices as components into other microservice projects |
| **Environments** | Environments get added by dev ops engineers. The developed dataflow projects get deployed to these added environments |
| **DataTransferObjects** | The data trnansfer objects are used inside of microservices by developers to help model the dataflow |

Table ‑ Overview of domain model classes

## Data Transfer Objects

The point of using a dataflow is to model an actual developed application as closely as possible. During the initial design and mockups, it was identified that there was a need to be able to pass around data from one dataflow component to another, much in the same way as DTOs do in Object Oriented Programming Languages. To this end it was identified that a way to build DataTransferObjects and pass them around in the dataflow components would be built.



# Class Diagram

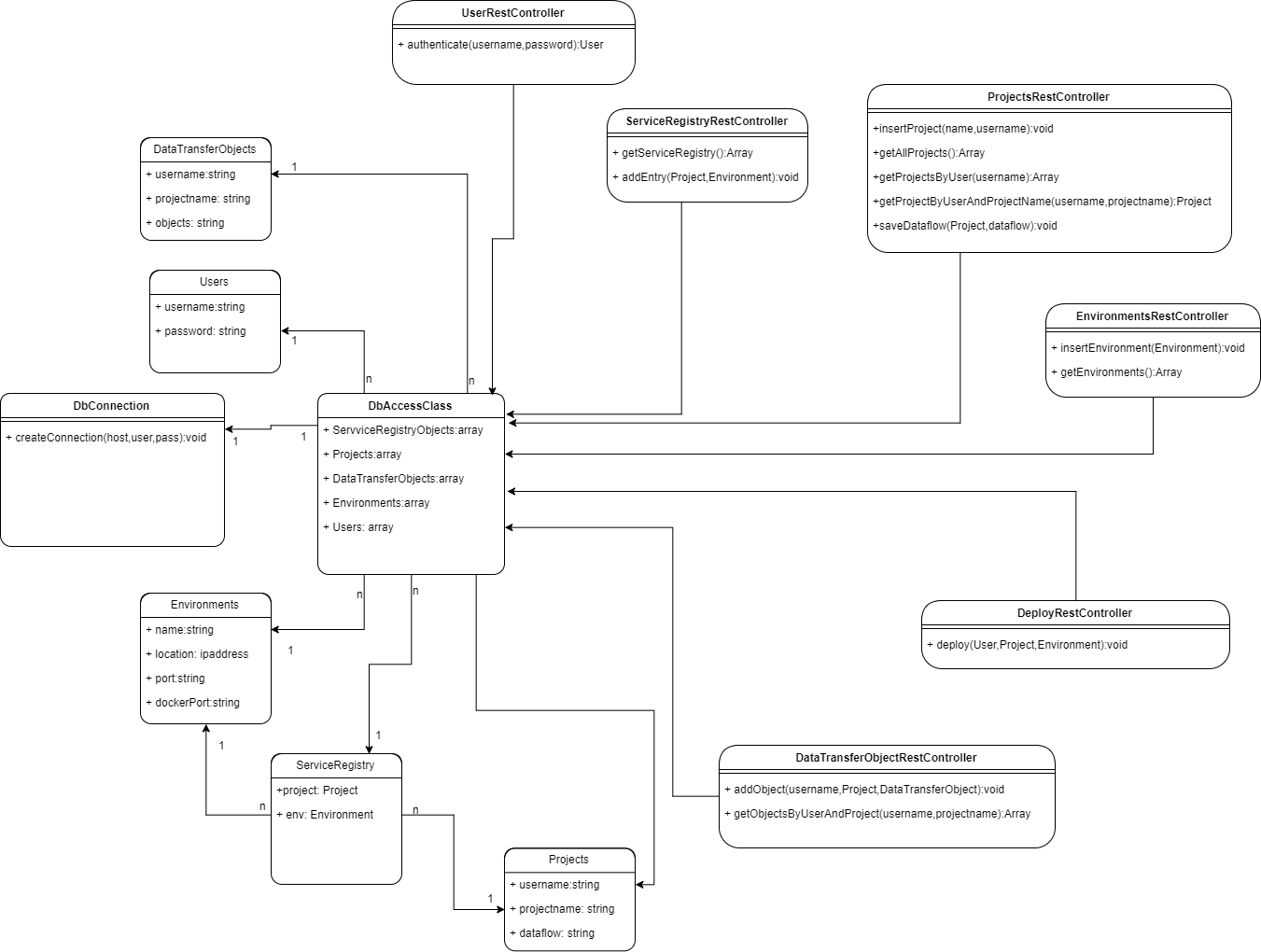


Figure ‑ Class Diagram

|  |  |
| --- | --- |
| **Class** | Description |
| **Users** | All users (developers, dev ops engineers, etc) are modelled through this class |
| **Projects** | Developers create projects. These projects house the dataflow diagram, and deployment environments. |
| **ServiceRegistry** | All deployed microservices get added to the service registry, to allow developers to add these microservices as components into other microservice projects |
| **Environments** | Environments get added by dev ops engineers. The developed dataflow projects get deployed to these added environments |
| **DataTransferObjects** | The data trnansfer objects are used inside of microservices by developers to help model the dataflow |
| **DbConnection** | The class that creates and connects to the DB |
| **DbAccessClass** | This class models the DB and contains read and write methods for all Db object classes |
| **UserRestController** | This controller models the REST API for developers to authenticate into the platform |
| **ServiceRegistryRestController** | This controller models the REST API for microservices to query the registry for microservice components |
| **ProjectsRestControlleer** | This controller models the REST API for developers to create and query projects |
| **EnvironmentsRestController** | This controller models the REST API for developers to add and query environments |
| **DeployRestController** | This controller models the REST API that allows developers to deploy the microservice into a Docker container |
| **DatTransferObjectRestController** | This controller models the REST API for developers to create a query DataTransferObjects related to each project |

Table ‑ Class Diagram Explanation

# Sequence Diagram

**Login System**

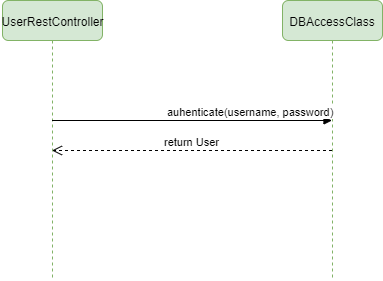


Figure ‑ Sequence diagram for login

A developer logs into the microservice platform. If the login details are correct the User Object is returned. Upon going to the home page after login, the GUI requests the details of the projects and environments that are related to the user and displays them on the GUI as appropriate.

**Get Projects**

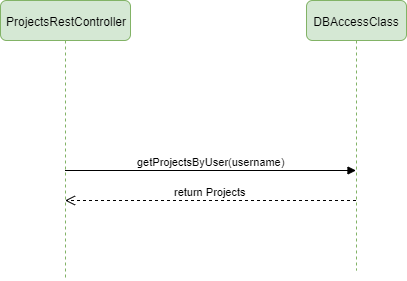


Figure ‑ Get Projects

Projects are returned by username after the developer has logged in.

**Get Environments**

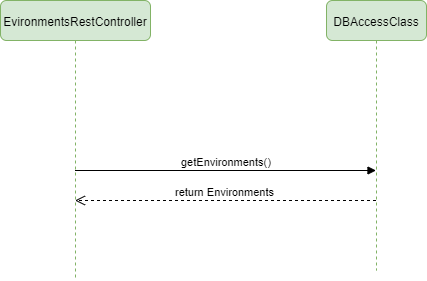
****

Figure ‑ Get Environments

Environments are returned by username after the developer has logged in.

**Create Environment**

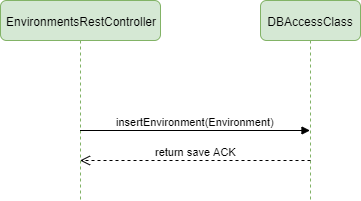


Figure ‑ Sequence Diagram for Add Environment

A developer uses the add button to open up the add environment dialog and enter the environment details. These details are then saved in the database.

**Create Project**

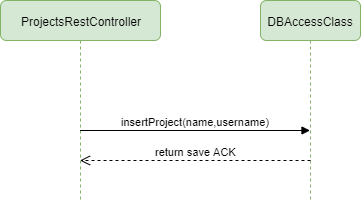


Figure ‑ Sequence Diagram of Add Project

A developer uses the add button to open up the add project dialog and enters the project details. These details are then saved in the database.

**Create Dataflow**

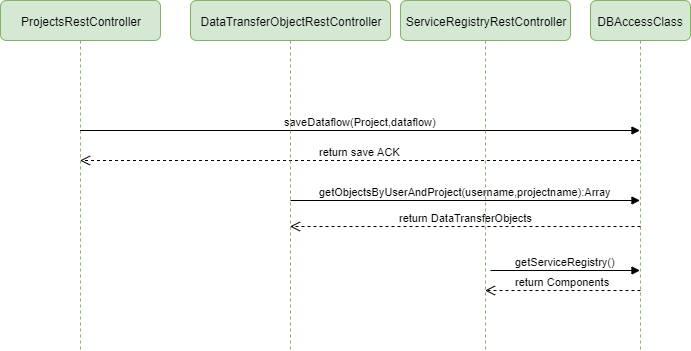


Figure ‑ Sequence Diagram of Creating a dataflow

The developer selects the project to open the dataflow tool. This activates a sequence of actions:

1. If a dataflow already exists it is returned and populated into the dataflow tool
2. DataTransferObjects related to the project are returned from the DB and populated in the dataflow tool
3. The Service Registry is queried to return other microservices to turn into microservice components that can be used in the dataflow.

The developer next spends time creating the dataflow itself using the available components.

**Save Dataflow**

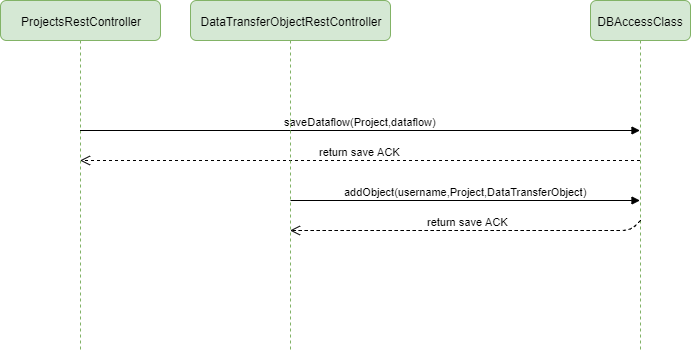


Figure ‑ Save Dataflow

Once the dataflow is finished being created the developer saves it. This activates two actions:

1. The dataflow itself gets saved to the DB
2. DataTransferObjects that the developer created get saved to the DB

**Deploy Microservice**

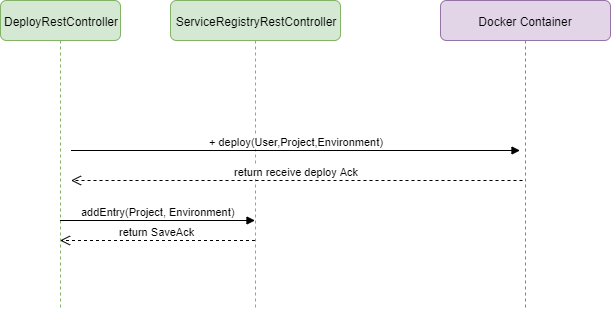
****

Figure ‑ Deploy Microservice

Once everything is ready the developer deploys the microservice. This sparks a sequence of actions in the following order:

1. The Environment details are returned from the DB based on the developer’s selection
2. The dataflow combined with the DataTransferObjects are transpiled into a NodeJS app
3. This app is sent to the relevant server based on the environment details
4. There it is compiled into a Docker container and deployed
5. An entry is added into the service registry stating that the application has been deployed. This entry will be queried in later microservices to be used as a microservice component
6. The developer receives an acknowledgement that the microservice has been deployed.

# Dataflow Tool

The below figure represents a mock-up of what a dataflow diagram should look like once built.

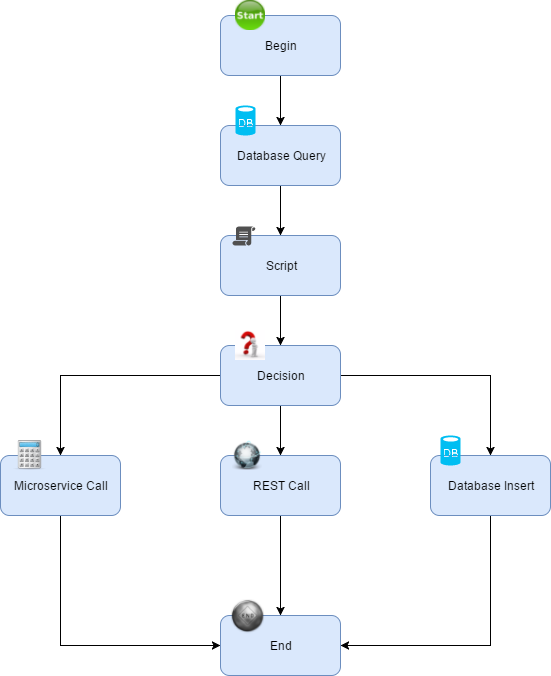


Figure ‑ Dataflow Mockup

The purpose of using a visual flow as opposed to a programming language is to make it easier to develop a microservice. This requirement was identified early on during the literature review and requirements specifications phase.

Most Object Oriented programming languages have a lot of common features and syntaxes that allow them to be used:

Using the Java language specification(Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad; Buckley 2014) as a base, several features of programming languages have been identified and modelled into the dataflow creation tool:

* Ability to define variables and data transfer objects. The supported types are here:
  + Integer
  + String
  + Double
  + Float
  + Arrays
  + Complex Objects
* If and switch condition handling
* Recursion
* Methods

## Dataflow Tool Components

In order to be able to model the above identified functionality as well as the ability to build functional applications a series of components has been introduced into the dataflow builder.

### Begin Component

This component signifies the beginning of the microservice. This component houses two major features:

* The ability to define variables, arrays, and complex objects which are saved in the DataTransferObjects class according to the domain model. These objects get used throughout the dataflow and consequently the microservice as well once its language equivalent has been compiled.
* This component also allows building of data objects that act as initial input by the microservice. These objects are exposed to the outside world by the microservice as a swagger definition.

### Database Query Component

This component is the first of many components that house complex functionality. Since the aim of the dataflow creation tool is to make development of microservices easier, several components that simplify complex functionality has been added.

The database component acts as an abstraction layer, to communicate between databases; querying data and saving it into a data transfer object.

### Script Component

The script component allows developers to embed Javascript into the microservice. This component was introduced to allow developers model complex business logic without a hassle.

When deployed the microservice evaluates the Javascript and executes the functionality.

### Decision Component

This component was introduced to allow users to make forks and decisions in the dataflow based on previous actions taken. A value in a variable or a data object is evaluated; based on the results a path is selected for the microservice to take.

### Database Insert Component

The component serves as the counterpart to the database query component, allowing developers to insert or update data into a database.

### REST Call Component

This component also houses complex functionality. This component allows developers to do REST calls to external servers and get their output. This output gets embedded into a data transfer object for use further down the dataflow.

### Microservice Component

This component is what is used to capitalize on the strength of microservices. The very philosophy of using microservices is the ability to string a set of microservices; passing data into, out of, and between the microservices before giving the output to the microservice caller.

The aim of this component is to emulate and model this philosophy. After connection, the dataflow tool calls the service registry and gets the swagger spec of the microservice to get the input and output data. The component allows the input data to be filled by the current defined variables, and the output to fill the rest of the defined variables. When compiled and placed into the Docker container, this component will call the external microservice, getting the location data from the registry.

### End Component

The final component that signifies the end of the microservice, this component gives the output of the microservice as a swagger definition that can be accessed via the service registry by other microservices, when the microservice component has been dragged into another project.

# Microservice Platform

The below diagram gives a component level view of the microservice platform.

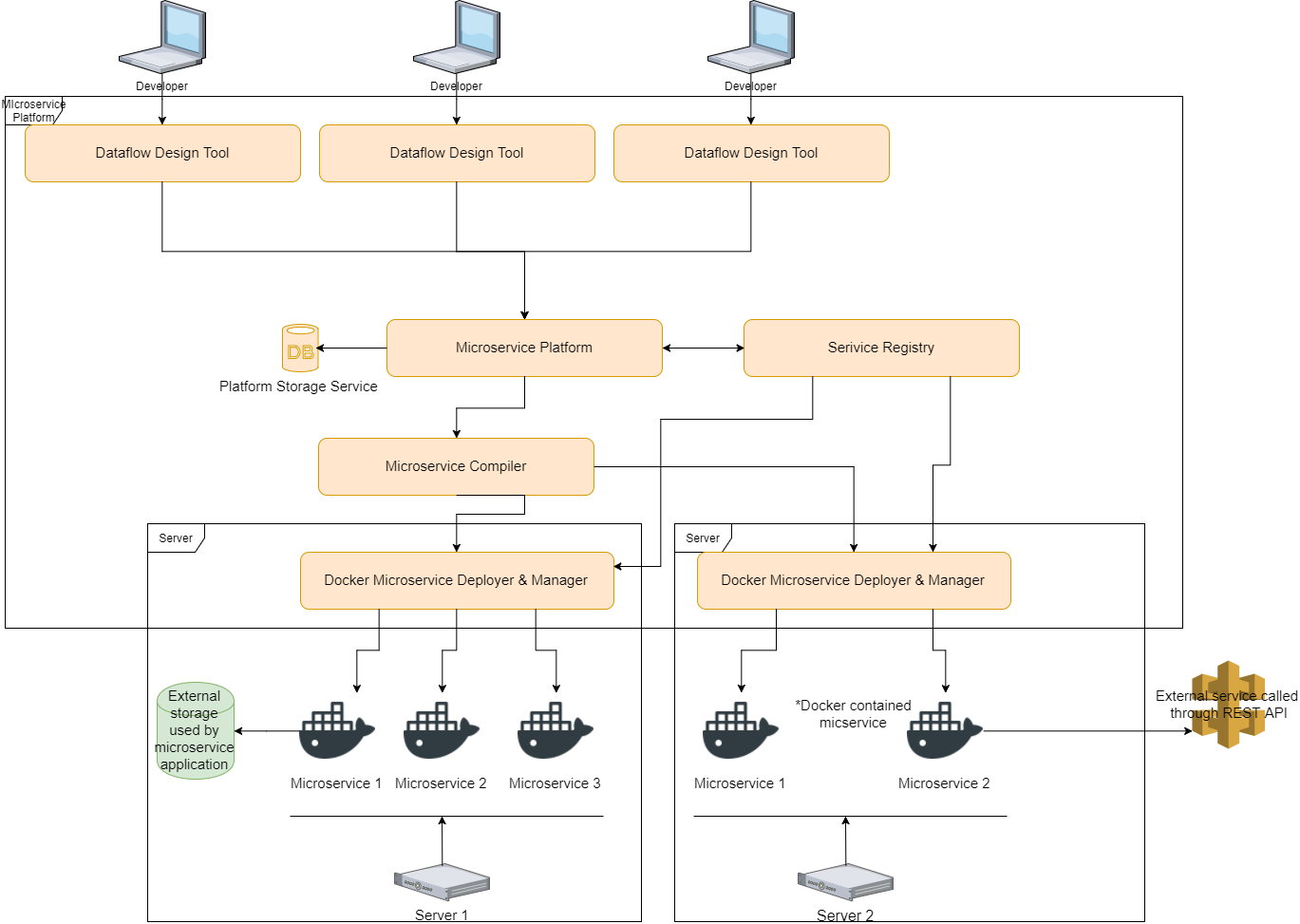


Figure ‑ Component Level View of Microservice Platform

### Docker Container

The end result of using the microservice platform should be a working microservice setup and running for you. It was concluded in the literature review that the best way to host the microservice is through a Docker container. The dataflow is first transpiled (translated from a dataflow to NodeJS and compiled) into a NodeJS app through the microservice compiler. This app is pulled into a NodeJS based Docker container and hosted on a server. The NodeJS app that is the microservice inside the container exposes a single REST API endpoint that was transpiled from the dataflow’s Begin and End components. The location of the microservice is accessible from the service registry and a swagger specification is exposed through the REST API for other developers to view or other microservices to use.

### Docker Deployer and Microservice Manager

While this component is part of the Microservice platform it actually resides on the hosting server’s environment. Its function is to actually deploy the Docker container. Deploying the container requires several steps and Linux terminal commands; as well as the creation and use of a Dockerfile (which is part of the Docker container). These commands are fired through this component and the Dockerfile is created through a FileWriter. The Dockerfile itself takes care of placing the NodeJS app inside the container.

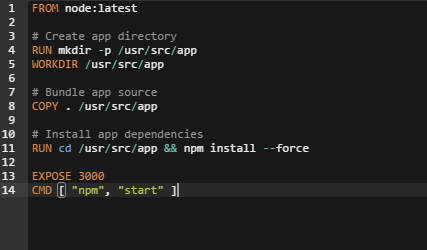
The below screenshot shows a sample Dockerfile 

Figure ‑ Dockerfile Screenshot

### Microservice Compiler

The microservice compiler is the component that actually transpiles the dataflow into a NodeJS app. A piece of template JavaScript code is kept inside the component; these template pieces are put together according to the flow of the diagram and compiled into a NodeJS app.

### Service Registry

The original need for a service registry was first identified during the Literature Review phase. Essentially this component acts as a repository for and of all existing microservices, their configurations and their hosted details. The main function of this component is to enable API Discovery which is one of the key requirements identified during the Requirements specification phase. This dataflow tool possesses the ability to drag another microservice as a component; this functionality is enabled by the service registry which allows searching and passing the details of a microservice to the dataflow tool, which turns these details into a dataflow component for developers to use.

### Microservice Platform

This is the central management component of the entire platform. This component exposes a REST API, which allows developers to sign in, create and save projects and acts as an interface to the storage layer of the microservice platform. In addition, its second major functionality is the central location; enabling the interfacing and communication all other components through this central hub.

# Chapter Summary

The chapter started by looking at the rich picture of the design of the system. It then gave an overview of the high level architecture and described its components. After that a Design Methodology was selected to be followed and the Domain Model of the design was discussed. This chapter ended by breaking down the two major parts of the prototype were broken down to low level components and discussed in detail. The next chapter discusses the implementation of these design decisions.

# System Implementation

Contents

* Chapter Overview
* Programming Languages
* Choice Of IDEs
* Coding Standard
* External Libraries
* Prototype Features
* Chapter Summary



# Chapter Overview

The chapter examines the implementation details of the prototype. First the choice of programming languages is presented, exploring the choice of JavaScript and NodeJS. Secondly the IDEs used in the development are explored and justified. Thirdly the chapter looks at the coding standards followed in the implemented codebase. Fourthly the external libraries used in the development process are identified and defended. The chapter concludes with a discussion of the implementation of the key features of the prototype.

# Programming Languages

The programming languages used in the development of the prototype are discussed.

### Javascript

The project involves the use of JavaScript for both the client side as well as the server side. JavaScript’s evolution as a language has recently brought many exciting advances to the world like NodeJS(Linux Foundation n.d.), AngularJS(Google Inc n.d.), ReactJS (Inc n.d.) as well as a host of supporting JS libraries which can be found and accessed easily from Github and NPM.

Although a number of JavaScript engines such as Rhino and V8 exist in the market, all implementations adhere to the ECMA standard. Thus JavaScript is interoperable and guaranteed to behave consistently. In addition, a plethora of tutorials and resources exist which make it accessible to most web application developers.

#### JavaScript is an Object Oriented Programming Language

Douglas Crockford, one of the developers of JavaScript and one who popularised the use of JSON, explains in his tutorial(Crockford n.d.) how JavaScript is an object oriented language. Most confusion stems from the fact that JavaScript is not Class based, which most inexperienced developers associate with being the only way to create an Object Oriented Language. Instead JavaScript is a Prototype based language which can be used to develop using the concepts of Object Orientation as shown in his tutorials.

### XML

XML(W3 n.d.) will be used to store the dataflow that is created by the developer. The graphing library that is used to draw the diagram will generate this XML string.

Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.

XML is human readable and is accessible to a large base of web application developers due to the readily available tutorials. In addition, XML has a number of libraries which allow files to be easily serialized and deserialized.

### JSON

JSON(ECMA n.d.) is an alternative to XML, offering more compact, easy parsers and inbuilt support in JavaScript. It is also very popular for data transfer in REST calls. Additionally Swagger supports a JSON parser, and most NoSQL databases can take in JSON objects without much modification.

# Choice of IDEs

Two IDEs were selected for the development of the prototype:

* C9(C9 n.d.)
* Brackets(Adobe n.d.)

C9: is a rich and comprehensive online IDE providing support for many languages, and being able to not only compile but also execute applications allowing them to be accessed to online. The main advantage is its portability, which allows it to be accessed anywhere online as long as there is an internet connection. This also allows for collaborative development which is also another major feature of this IDE.

Brackets: is an IDE developed by JavaScript for JavaScript. Although lightweight, it is as fully featured as most heavy-duty IDEs. For times when portability is not required this IDE is used to do development on the prototype.

A JavaScript IDE must have code assist (code completion), syntax highlighting as well as been able identify dependencies in AMD modules. The choices hinged between Notepad++(Notepad++ n.d.), Visual Studio(Microsoft n.d.), Komodo(Komodo n.d.), Netbeans(Netbeans n.d.), the Eclipse(Eclipse n.d.) JavaScript development plugin which is bundled with the Web developer package and Brackets. Visual Studio provides JavaScript debugging support through integration with Internet Explorer. Although this is a valuable feature, the development activities focused around Chrome browser for which Visual Studio does not provide support. In addition, it is unable to .The Visual Studio option was discarded as it proved to be bulky requiring the installation of several Microsoft technologies to function. The Komodo IDE required a hefty monetary investment and thus was not feasible for a student research project. The Netbeans IDE offered code assist and syntax highlighting but it was noted to perform sluggishly on the development machine as noted earlier in the Java IDE comparison. The Eclipse IDE would have offered a convenient single solution for both Java and JavaScript development, but the JavaScript extension refused to function correctly. Notepad++ and Brackets are almost similar but Brackets provides more JavaScript centric features like live development and reload, therefore Brackets was chosen.

# Coding Standard

A consistent coding standard was adopted across the JavaScript code bases with minor changes to commenting styles. The coding style was adapted from the specification created by Williams and Isiah(Williams 2010).

## Variable and Method Naming

The naming of variables and functions follow the Camel casing style. This style was chosen due to its adoption as the defacto Javascript coding style(Williams 2010). As a result the code base should be familiar to all JavaScript developers.

## Class Names

The JavaScript classes begin with an upper case character with subsequent words beginning with upper case letters.

## Comments

The comments for the JavaScript code follow @ Purpose , @ Parameters

# External Libraries

The project utilized several external libraries to aid the development and testing.

## NodeJS

NodeJS is a development library used for the development of server side applications.

As an asynchronous event driven JavaScript runtime, Node is designed to build scalable network applications. Node is similar in design to, and influenced by, systems like Ruby's [Event Machine](http://rubyeventmachine.com/) or Python's [Twisted](http://twistedmatrix.com/). Node takes the event model a bit further. It presents an [event loop](https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/) as a runtime construct instead of as a library. In other systems there is always a blocking call to start the event-loop. Typically behavior is defined through callbacks at the beginning of a script and at the end starts a server through a blocking call like *EventMachine::run()*. In Node there is no such start-the-event-loop call. Node simply enters the event loop after executing the input script. Node exits the event loop when there are no more callbacks to perform. This behavior is like browser JavaScript — the event loop is hidden from the user.

HTTP is a first class citizen in Node, designed with streaming and low latency in mind. This makes Node well suited for the foundation of a web library or framework.

Just because Node is designed without threads, doesn't mean you cannot take advantage of multiple cores in your environment. Child processes can be spawned by using [*child\_process.fork()*](https://nodejs.org/api/child_process.html#child_process_child_process_fork_modulepath_args_options) API, and are designed to be easy to communicate with. Built upon that same interface is the [cluster](https://nodejs.org/api/cluster.html) module, which allows you to share sockets between processes to enable load balancing over your cores.

The microservice platform’s server side will be built using NodeJS. One of the main advantages of NodeJS is the availability of a myriad of libraries for almost any type of function due to its popularity.

## AngularJS

The front-end of the microservice platform which the developers will see and use will be built in AngularJS(Google Inc n.d.) The library was used to make frontend development easier.

AngularJS is what HTML would have been, had it been designed for building web-apps. Declarative templates with data-binding, MVW, MVVM, MVC, dependency injection and great testability story all implemented with pure client-side JavaScript.

## mxGraph

mxGraph(jgraph n.d.) is an open-source JavaScript graphing library, which contains a comprehensive list of features. Modifications to the library will be done to allow developers to use the generated UI like a MDA tool to draw the dataflow of the microservice. The library will be used inside of AngularJS to allow developers to build their dataflows.

# Prototype Features

The prototype was developed as two sections as discussed in the Design Chapter:

* Dataflow Tool
* Microservice Platform

## Dataflow tool

The dataflow tool was implemented using a combination of AngularJS and the mxGraph library on the frontend and NodeJS and MongoDB(MongoDB inc n.d.) as the database.

MongoDB is document storage database which is different from the usual SQL related relational databases. The reason MongoDB was chosen over a relational DB is due to high compatibility with JavaScript frameworks. A JavaScript or JSON object can be directly passed into and queried from the DB without the need for pesky SQL queries and relational structuring. This makes development much faster and given the time constraints a much more suitable choice.

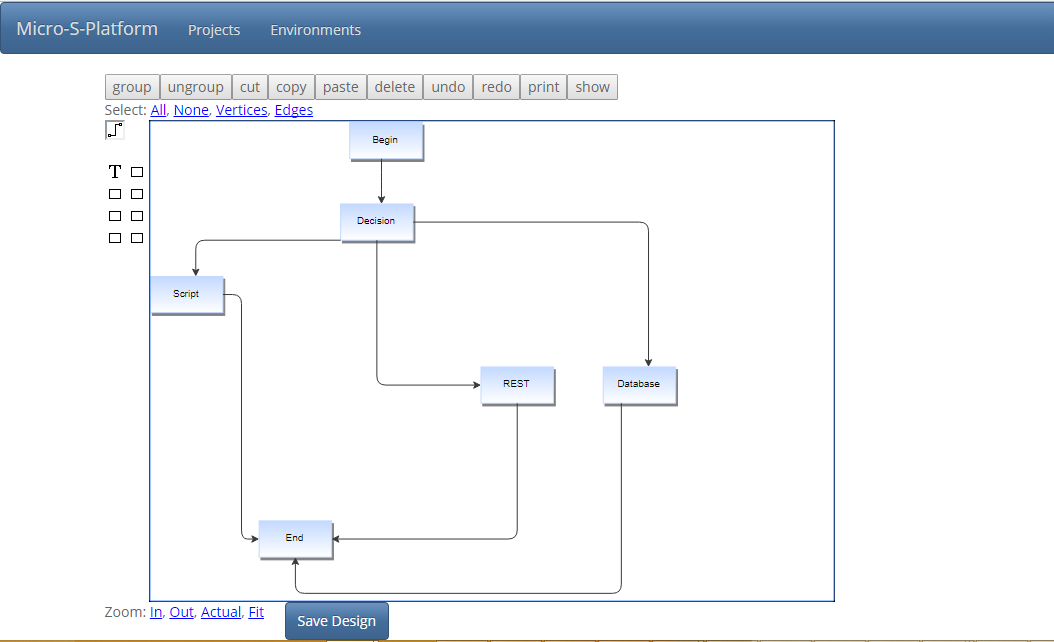


Figure ‑ Dataflow tool screenshot

The dataflow tool builds the flow of the microservice. This can be seen in the central canvas section in the screenshot above. The left side contains the usable list of components. All around the tool are various additional functionalities provided by the xmGraph library.

### Dataflow tool Output

The mxGraph generates the output from the library itself. Below is a provided sample

*<mxGraphModel>*

*<root>*

*<Diagram label="My Diagram" href="http://www.jgraph.com/" id="0">*

*<mxCell/>*

*</Diagram>*

*<Layer label="Default Layer" id="1">*

*<mxCell parent="0"/>*

*</Layer>*

*<Begin label="Begin" href="" id="2">*

*<mxCell vertex="1" parent="1">*

*<mxGeometry x="90" y="50" width="80" height="40" as="geometry"/>*

*</mxCell>*

*</Begin>*

*<Script label="Script" href="" id="3">*

*<mxCell vertex="1" parent="1">*

*<mxGeometry x="230" y="110" width="80" height="40" as="geometry"/>*

*</mxCell>*

*</Script>*

*<End label="End" href="" id="4">*

*<mxCell vertex="1" parent="1">*

*<mxGeometry x="350" y="150" width="80" height="40" as="geometry"/>*

*</mxCell>*

*</End>*

*<Connector label="" href="" id="5">*

*<mxCell edge="1" parent="1" source="2" target="3">*

*<mxGeometry relative="1" as="geometry"/>*

*</mxCell>*

*</Connector>*

*<Connector label="" href="" id="6">*

*<mxCell edge="1" parent="1" source="3" target="4">*

*<mxGeometry relative="1" as="geometry"/>*

*</mxCell>*

*</Connector>*

*</root>*

*</mxGraphModel>*

**Noteworthy Features**

**Component (*“Begin”*, *“Script”*, *“End”*):** The xml of this component is what handles the data related to the functionality of the component itself. The *“label”* and *“id”* attributes in particular are of use later to the transpiler.

**Connector:** The xml related to this configuration defines the connections between two components. Specifically the *“source”* and *“target”* get used by the transpiler during deployment.

## Microservice Platform

The first step when deploying the application is to take the generated xml and convert it into a working application. For this an existing NodeJS template app has been created.

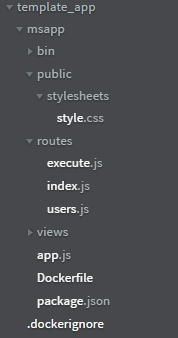


Figure ‑ Template App Folder Structure

NodeJS has the inbuilt ability to execute OS commands and since the Microservice Platform runs on a Linux OS; Linux terminal commands can be executed.

Below is the screenshot showing the OS commands used by the compiler.



Figure ‑ OS Commands

First a copy of the template application is created with the name of the project. Then the logic is compiled by the transpiler and placed into the copied app using an inbuilt FileReader and FileWriter found in NodeJS. Below shows the screenshot of where the compiled logic takes place.



Figure ‑ Template App Logic Location

Once the logic compiled and place into the copied app, all of it is compiled into a Docker container and set to run on the hosting machine.

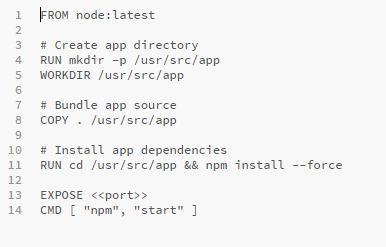


Figure ‑ Template App Docker

The deploying location is passed as the environment variable to the compiler. The compiler is what decides which ports to place the Docker container and NodeJS app with. After being deployed the statuses are checked by *“docker images”*and *“docker ps”* commands to see of the containers are up and running.

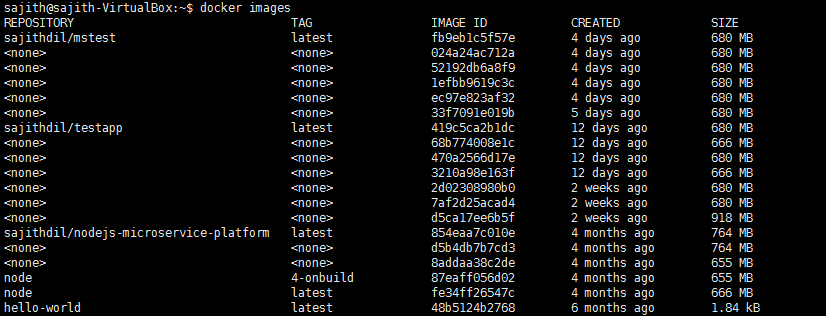


Figure ‑ *“docker images”* result

docker_p.PNG

Figure ‑ “*docker ps”* result

Once the Docker container’s integrity has been verified the deployment success ACK is sent back to the compiler to complete the deployment and send the success message to the user.

# Chapter Summary

The chapter examined the implementation of the design defined in the Design chapter. The choice of tools and languages was justified in terms of the development considerations of the project. The coding standards for the code base were identified and related to an official standard. The chapter then examined implementations of the key features and provided details on specifics of each feature. The next chapter examines provides an evaluation of the prototype along with the results of testing.

# System Testing

Contents

* Chapter Overview
* Objectives and goals for Testing
* Testing Process
* Testing Criteria
* Adopted Testing Method for Functional Requirements
* Areas Tested – Functional Requirements
* Areas Tested – Non Functional
* Chapter Summary



# Chapter Overview

Having discussed about the implementation of the proposed prototype in the last chapter, this chapter will be on testing of the functional and non-functional requirements of the implemented system with the intension of making certain that all the implemented requirements were completed to the expected levels. This chapter first examines the testing process used to give structure to the testing process, the testing criteria, testing methods and testing levels and finally will provide an evaluation of the testing results.

# Objectives and goals for Testing

Software testing is performed to verify that the completed software package functions according to the expectations defined by the requirements.

The main objectives of the testing process for the Microservice Platform are:

* To verify and validate the functional requirements of the Microservice Platform system.
* To verify and validate the non functional requirements of the Microservice Platform system
* To identify the errors and defects of the system in order to make sure that the final product contains fewer amounts of errors and bugs.
* To further enhance the system based on the test results.

# Testing Process

Early on in the lifecycle of the project it was necessary to breakup testing into focusing on the dataflow tool and the transpiler, as the microservice platform required a compilable NodeJS application that executed on at least a happy path, that could be placed in a Docker container so that platform level testing could occur. The test plan was thus driven by the need to test these components from requirements gathering to implementation. The test process was modeled after the BS 7925-2(British Standards Institute 1998) software component testing standard given in the figure below

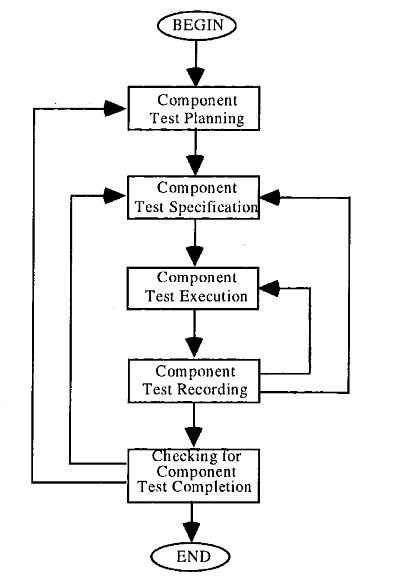


Figure ‑ Component Testing Flow

Since the dataflow is made up of components this presents an ideal method to test each component individually early on in the development lifecycle. However due to the nature of the project it was not possible to follow the above generic process for all the components of the prototype

# Testing Criteria

Testing the implemented system is the process of an examining an application to ensure that it satisfies the functional and non-functional requirements and meets quality expectations. Software quality can be measured in two ways as described below

1. **Software functional quality**

Mainly focus on the combination of the product development characteristics with the technical requirements of the given design based on the functional requirements.

1. **Software structural quality**

This uses to measure the performance of the functional requirements of the product with the identified non-functional requirements.

.

# Adopted testing method for functional requirement

Spiral methodology(Boehm 1988), which is the adopted software development methodology, allows the flexibility and freedom of carrying out the testing of the software parallel with the implementation phase. Hence the testing of the implemented functional requirements was carried out parallel to the implementation of the prototype system with the black box testing approach.

# Areas Tested – Functional Requirements

The areas of testing extended into several components at a quantitative and qualitative level.

1. Dataflow tool
2. Dataflow tool components
3. Microservice Platform
4. Transpiler (Microservice Compiler)
5. DataTransferObjects
6. Service Registry
7. Swagger Output
8. Docker Container Integrity Verification

## Dataflow Tool and Microservice Platform

Testing of the tool itself requires not only testing its functionality but its browser compatibility as well since the tool itself is web based. The microservice platform functionality was tested with this as well because platform is what returns the data to be displayed by the tool.

The following list of browsers were considered for testing

* Chrome
* Firefox
* Opera
* Safari
* Microsoft Edge
* Internet Explorer(IE) 7
* IE 8
* IE 9
* IE 10
* IE 11

Mobile browsers were not considered in the testing cases as developers generally do not develop on mobiles. However developers do develop on tablet pc. These test cases were covered inside the above browsers since the same browser engines are used on PC and tablets for easier compatibility.

The full test cases for the dataflow tool itself can be found in Appendix B under the section Dataflow Tool Testing.

|  |  |  |  |
| --- | --- | --- | --- |
| Case No | Description | Pass Rate | Pass Rate without IE |
| 1 | Login to System | 50% | 100% |
| 2 | Add Project | 50% | 100% |
| 3 | Add Environment | 50% | 100% |
| 4 | Open Dataflow tool | 50% | 100% |

Table ‑ Dataflow Tool Testing Result Summary

The above table summarizes the results of the functional test on various browsers. The pass rate was calculated using the following formula:

Equation ‑ Pass rate calculation formula

The second pass rate was calculated using the same criteria as above but without considering IE related browsers

Equation ‑ Pass rate calculation formula without IE

Two separate pass rates needed to be considered. This is due to the fact that IE has compatibility issues(Google Inc n.d.) with AngularJS. These compatibility issues were not considered during development of the prototype and were only made apparent during the testing functional testing phase of the lifecycle.

However testing passed 100% on browsers that were not IE related. IE compatibility is unlikely to be an issue as it is not a browser that is in common use as can be seen in the following diagram taken from(w3schools n.d.)

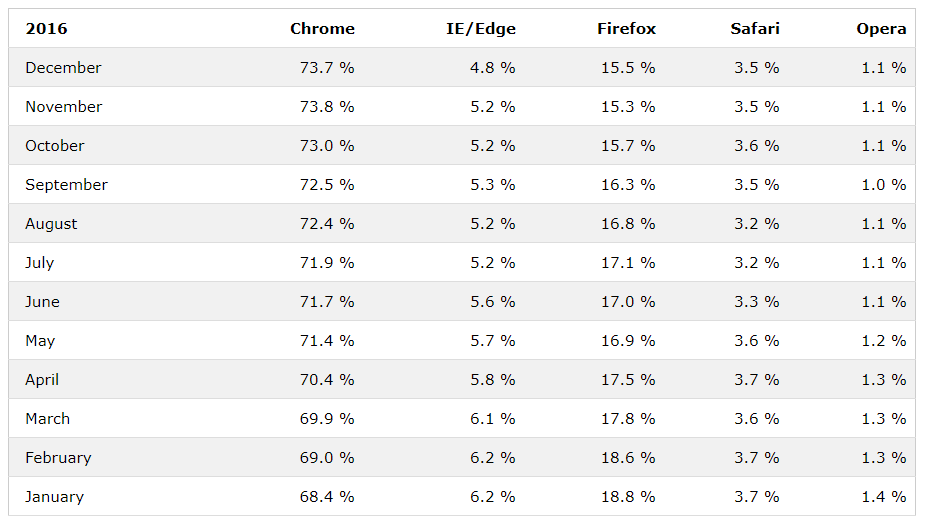


Table ‑ 2016 Browser usage statistics

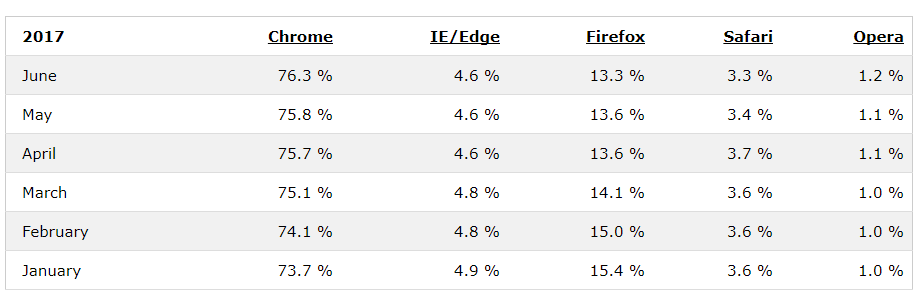


Table ‑ 2017 Browser Usage statistics

## Dataflow Tool components, DataTransferObjects and Transpiler

The components themselves possess several functionalities, and since the transpiler is what converts the GUI functionality into NodeJS code its best to test these together to see if each function gets compiled into the relevant piece of code correctly.

There is no need to test each of the components functionality on separate browsers since the component’s GUI functionality depends on the dataflow library itself and the library functionality opening has already been tested previously. The compiler’s functionality itself can be tested on the server side by opening the created NodeJS application and inspecting the code. Since the code itself is JavaScript there is no need to be decompile it in any way, it’s already in human readable format.

|  |  |
| --- | --- |
| **Component Name** | Test Cases Handling |
| **Begin Component** | The Begin component’s two major functionalities are tested separately and can be found in Appendix B under Begin Component testing. While functionalities are different of both features, its end result is the same: a variable needs to be created in the end. As such similar test cases can be applied to both features to allow testing without needing to come up with a new set of test cases. |
| **Database Query Component** | This component’s basic functionality is tested by passing data from DataTransferObjects to see if the data returned is correct. Therefore all basic variable types will be tested, but complex objects and arrays will be skipped as they do not apply to database queries. The test cases and pass status can be found in appendix B under Database Query Component Testing |
| **Script Component** | This component’s testing encompasses a large quantity of features. Not only will testing to check whether the DataTransferObjects can be accessed will need to be done, but generic programming concepts such as internal variable creation, loops, switch, and if conditions have to be tested here as well. The testing for this component can be found in appendix B under Script Component Testing. Fully testing programming concepts of this component may be infeasible using the method of black box testing since a developer is unable to see its internal workings. White box testing here is a better approach since the internal workings of the code can be read (this is how the transpiler’s functionality is evaluated). |
| **Decision Component** | Testing for this component will evaluate DataTransferObject loading and checking against both other DataTransferObject values and hardcoded values under various conditions The testing for this component can be found in appendix B under Decision Component Testing. |
| **Database Insert Component** | Testing similar to the Database Query component will take place here as well. Testing can be found under the Database Insert Component Testing section in Appendix B |
| **REST Component** | The four major REST methods (PUT, POST, GET, DELETE) will be tested in combination with DataTransferObject access testing. An external sample application will be written to allow for testing |
| **Microservice Component** | This testing is done last after ensuring all other components are in working order. This is to ensure that a working application can be compiled and deployed so that a component can be created and accessed. Most testing will be done to see if DataTransferObjects are accessed correctly for the GUI functionality. The rest of the functionality will be tested during the Service Registry testing. |
| **End Component** | This component testing involves testing to see if the output objects parse to swagger correctly. Since this component is similar to the Begin Component’s Swagger objects input feature, the same test cases can be used here as well for testing. The test cases and status can be found in Appendix B under End Component Testing |

Table ‑ Componenet Test Case Handling

## Swagger Output Testing

Swagger output testing was done during the testing of the Begin and End Component.

## Service Registry Testing

When the microservice deploys an entry is added to the registry. This registry is called when loading the dataflow tool to create microservice components. This is simple functionality testing and does not require complex test cases to test.

## Docker Container Testing

Since microservices are deployed to Docker containers, the integrity and existence of the container must be verified. This was done during the microservice component testing since the component calls the NodeJS application inside the Docker container. If everything inside the Docker container and the container itself is running correctly then the Microservice component should have got its output correctly. Therefore this feature does not need a separate test plan; its functionality was tested during the testing of the microservice component.

## Functional Requirements Met

|  |  |
| --- | --- |
| **Functional Requirement** | **Status** |
| Developers should be able to use the dataflow tool to drag and drop components to develop the microservice | Pass |
| Developers should be able to use the microservice platform to deploy the microservice to different environments | Pass |
| The microservice platform should deploy the microservice into a Docker container | Pass |
| The microservice platform should posses a service registry to handle communication | Pass |
| The microservice platform should use Swagger to expose the metadata of the input and output of microservice | Pass |
| The service registry should be able to manage multiple environments for each microservice | Pass |
| Dev Ops should be able to enter configuration details of different environments for each microservice | Pass |
| The microservice platform should be able to deploy to multiple platforms simultaneously | Pass |
| It should be able for a developer to discover another microservice and use as a component in his microservice using the service registry | Pass |

Table ‑ Functional Requirements Testcases

# Areas Tested – Non Functional

Several Non functional requirements were identified during the System Requirements Specification:

|  |  |
| --- | --- |
| NFR 1 | Usability |
| The system should ensure higher level of usability by making the prototype usable for new users with minimum training. The user interface has to be simple and visually appealing to use. Help messages will be provided where necessary to improve usability. A new user should take maximum of 1 day to get familiar with the functionalities and the capabilities of the proposed system. | |
| NFR2 | Scalability |
| Another major non-functional requirement of the proposed system is to maintain a higher level of scalability. It’s expected that the workload of the system will increase as time goes on with the number of services increasing and the number of API consumers increasing as well. So the system should be scalable enough to handle to increasing workload in a satisfactory manner. | |
| NFR3 | Performance |
| System should respond quickly for the API consumers and work properly without crashing on all types of devices and platforms. | |
| NFR4 | Uptime |
| The system should be available for as long as possible. API consumers that use services will fail should the proposed system go down. Therefore keeping the system up and running for as long as possible without any downtime is mission critical. | |

**Usability**

Usability testing was done by giving the tool to a set of developers to familiarize with and try to use on their own. Below are the results of the usability test.

Figure ‑ Results of how long developers took to familiarize with the system

**Scalability**

Scalability was met early on during the design phase by implementing the concept of environments allowing for infinite scalability.

**Performance**

Performance testing is out of the scope of the prototype as performance is based on the Docker container and the hosting OS.

**Uptime**

Once the integrity of the Docker container has been verified, the uptime of the container is taken out of the prototype’s hands. Uptime depends again on the Docker container itself and the host OS.

# Chapter Summary

This Chapter focused on the testing aspect of the prototype and the chapter started with outlining the purpose of the testing phase, testing criteria, testing methods and testing levels. There were two main types of testing methods Software functional quality testing and software structural quality testing. Under software functional quality testing functional requirement testing and unit and integrating testing were carried out using the black box testing approach. The testing results emphasized that the functional and unit and integration of the system was completed to the expected level. Then the testing moved on to software structural quality testing which mainly focuses on testing the identified non-functional requirements. Next chapter which is the evaluation chapter will describe the evaluation process carried out on various evaluation criteria of the Microservice Platform project.

# Evaluation

Contents

* Chapter Overview
* Evaluation Criteria
* Selection Of Evaluators
* Evaluation of Methodology and Approach
* Summary of Evaluation of User Questions
* Evaluation Of Survey Results
* Self Evaluation
* Chapter Summary



# Chapter Overview

Previous chapter discussed about the testing process carried out on the prototype system with the intension of making certain that all the implemented functional and non-functional requirements were achieved to the expected levels. This chapter focuses on the evaluation process carried with various evaluators on different evaluation criteria and the self-evaluation in order to find out strengths and weakness of the Microservice Platform project.

# Evaluation Criteria

The following criteria were identified for the evaluation process and these criteria were selected with the aim of covering the major phases of the whole project.

|  |  |
| --- | --- |
| **Criteria** | **Description& purpose** |
| Overall concept and whole project | It is believed that the Microservice Platform should be able to create a dataflow and deploy it as a microservice |
| Scope and depth of the project | Since the domain itself is broad but still in its infancey, it is important to get views and comments about the scope of the project from domain experts |
| System design, architecture and implementation | Evaluate whether the design, architecture and implementation of each module is completed properly providing proper justifications. |
| Solution and Prototype | Assessment should be done on the prototype to determine whether the prototype acts as a proof of concept of the Microservice Platform project. |
| Usability, performance and accuracy of the prototype | Evaluate the non-functional requirements of the Microservice Platform to determine the extent the non-functional requirements were implemented. |
| Limitations of the solution and future enhancements | Identification of limitations of the Microservice Platform system and potential future enhancements that needs to addressed. |

Table ‑ Evaluation Criteria

# Selection of Evaluators

Below mentioned evaluator categories were identified for the evaluation process of the project and high priority will be given to the domain experts of microservices systems as a high emphasis of this project was given to the scope of microservices and end users as they would be the group ultimately using the developed system.

|  |  |
| --- | --- |
| **Evaluator Category** | **Description & criteria evaluated** |
| End users | Group of microservice and non microservice developers were selected for the evaluation of the overall concept, usability, performance and accuracy, prototype and future enhancements criteria of the Microservice Platform system. |
| Microservice Domain Experts | Group of experts who are well experienced and currently working in the microservice domain and on projects related to microservices were selected to evaluate the overall concept, prototype and future enhancement criteria |
| Software engineers & architects | Group of well experienced and currently working software engineers and architects were selected to evaluate the System design, Architecture and implementation and future enhancement criteria |

Table ‑ Evaluators

# Evaluation Methodology and Approach

Evaluation process of a project exposes the success of the project and provides a feedback with regards to implementation and major phases of the project life cycle such as problem, analysis, design and implementation. The project was evaluated using a combination of qualitative and quantitative methods. The quantitative evaluation is more or less achieved through the testing phase of the project. Hence more emphasis was given to the qualitative evaluation of the project this section. Evaluation of the Microservice Platform project was carried out through questionnaire and interview approaches, but major emphasis was given to questionnaire approach due to time constraints faced at the end of the project. One questionnaire was prepared on the qualitative measures of the dataflow tool, and distributed to experts in the field along with the source code, design and the implementation chapters. Another questionnaire was prepared on the qualitative measures of the design, architecture and implementation of the prototype and was distributed among the software engineers and architects. An additional questionnaire was prepared on the qualitative and quantitative measures of the non-functional requirements of the Microservice Platform system and overall concept of the project and was distributed among the end users of the Microservice Platform system. An Interview process was carried out with the domain experts to evaluate the qualitative aspect of the overall concept, scope as well as the prototype.

# Summary of Evaluation of Survey Questions

|  |  |
| --- | --- |
| No | Question |
| **User Related Questions** | |
| 1 | How many years of experience do you have in the microservice domain? |
| **The project concept and the project as a whole** | |
| 3 | What is your general idea about the Microservice Platform project? |
| 4 | What would be the impact this solution would have on the domain as a whole? |
| **Scope and depth of the project** | |
| 5 | Do you think the scope of the project is acceptable for postgraduate level? |
| 6 | If not, What depth of the solution should have addressed the microservice domain? |
| **System design, architecture and implementation** | |
| 7 | What are your comments about the design and architecture with regards to the project concept? |
| 8 | Do you think the decisions made in the implementation phase are acceptable and justifiable? |
| 9 | What are your suggestions on the design and the implementation? |
| **The solution and prototype** | |
| 10 | Do you think the presented solution is having the depth in solving the problem? |
| 11 | Do you think the system provides a solution to the identified problem? |
| 12 | What are your comments on the features offered? |
| **Usability, performance and accuracy of the prototype** | |
| 13 | How would you rate the usability, performance and accuracy of the prototype? |
| **Limitations of the solution and future enhancements** | |
| 16 | What are the general limitations you see in the solution and what are your recommendations for those? |
| 17 | What are the features do you think can be added to Microservice Platform project? |

Table ‑ Summary of questions

# Evaluation of Survey Results

The evaluation results are presented as a summary with some comments and suggestions of the evaluators.

## Overall Concept

*“The Proposed solution to address drawbacks of micro-service platforms by dataflow programming is outlined pragmatically and surrounds most of the academic aspects.  It is so fascinating to witness the prototype which has been implemented adapting to latest technology trends such as Virtualization, containers etc. Therefore, I recommend this dissertation as a comprehensive academic research for a Masters level.”*

Shashika Kodikara  
Manager – IT

Wavenet International (Pvt) Ltd

*“I think this is a very relevant subject area which you have explored. One major challenge that developers face is having to do rapid development and changes to meet customer expectations. And the answer that emerged was Micro-service architecture. So taking this theory and constructing it into a standardized tool is impressive”*

Chiranthana Weraniyagoda - BEng(Hons), MBA  
Head of Marketing

Wavenet International (Pvt) Ltd

*“This is really interesting sector which is emerging in cross platforms environments. Developing a fully functional system which is capable of satisfying nowadays requirement is highly time consuming and its contains much more issues when operating under multiple devices and platforms therefore to avoid these issue we can combine multiple platforms to gather and achieve our ultimate target by using a microservices platform.* *Accordingly I think this project has perfectly illustrated and explained all major functionalities and operation, Therefore I recommended this project as your academic research. “*

Prasad Jayasinghe  
Network Administrator – IT

Wavenet International (Pvt) Ltd

|  |  |
| --- | --- |
| Evaluation module | Summary of feedback |
| The concept | The feedback from the evaluators about the concept of the project was positive. Especially the general users were pleased to have a system like the Microservice Platform which would help in saving time and effort and they were highly satisfied that the same features were provided to the mobile platform as well. And domain experts were keen on the financial benefits this system would bring in to the domain. |
| Review on feedback | This presents a novel approach and a fresh look at the domain which was starting to reach a repetitive state with only rehashed solutions from other similar (monolithic, SOA, etc) domains. Most of the evaluators stated that this project has a good potential towards the success and were appreciative of the effort to add to the domain. |

Table ‑ Feedback Evaluation – Overall Concept

## Scope & depth of the project

*“Therefore, I recommend this dissertation as a comprehensive academic research for a Masters level.”*

Shashika Kodikara  
Manager – IT

*Therefore I recommended this project as your academic research. “*

Prasad Jayasinghe  
Network Administrator – IT

|  |  |
| --- | --- |
| Evaluation module | Summary of feedback |
| The scope | Overall comment was that the project scope is a challenging and the addressed depth of the project is sufficient enough for a post graduate level research. |
| Review on feedback | Microservices are a broad domain even though still in its infancy. With the feedback of the domain experts it can be concluded that the depth addressed in this research is sufficient enough for a post graduate level research. |
|  |  |

Table ‑ Feedback Evaluation – Scope and Depth

## System Design, Architecture and Implementation

*“It is so fascinating to witness the prototype which has been implemented adapting to latest technology trends such as Virtualization, containers etc”*

Shashika Kodikara  
Manager – IT

Wavenet International (Pvt) Ltd

*“Implementation of the 3-tier architecture based on NodeJS and the MEAN stack here is impressive”*

Chinthaka Gunarathne  
Head of Engineering

Wavenet International (Pvt) Ltd

|  |  |
| --- | --- |
| Evaluation module | Summary of feedback |
| System Design | Most of the reviewers were satisfied with the design methodology and design goals and decisions that were taken during the designing phase. Some criticized the use of SSADM for the designing of the data interaction writing module. |
| Review on feedback | The decision of using SSADM approach for the designing of the data interaction writing module was taken by considering the technologies involved in that module. Both NodeJS and Redis access layer supports structural programming and the best design methodology for structural programming is the SSADM approach. |
| Evaluation module | Summary of feedback |
| System architecture | Most reviewers praised the use of the three tier architecture and the use of NodeJS which allows it to be installed and used on most OS platforms easily. One architect even pointed out its easy extensibility to cloud hosting, but one other architect being partial towards SaaS (Software as a service) pointed out that this architecture is unsuitable towards it. |
| Review on feedback | The argument made that the current architecture would not support a SaaS platform is a valid point but the current architecture was derived for typical client server architecture and the most suitable architecture approach was chosen for the modelling of the system. If this system is going to be converted to a SaaS then the architecture of the system would have to be thought from the beginning. |
| Evaluation module | Summary of feedback |
| Implementation | Though most of the reviewers praised the technologies used for the implementation of the Microservice Platform system saying that those used technologies are light weight and cutting edge technologies, one software engineer specifically praised the use of Docker Containers. |
| Review on feedback | The technologies used for the implementation of the Microservice Platform system were reviewed for their advantages and limitations and the most suitable technologies were chosen. |

Table ‑ Feedback Evaluation - System Design, Architecture and Implementation

## Solution and Prototype

*“ It is so fascinating to witness the prototype which has been implemented adapting to latest technology trends such as Virtualization, containers etc”.*

Shashika Kodikara  
Manager – IT

Wavenet International (Pvt) Ltd

*“Developing a fully functional system which is capable of satisfying nowadays requirement is highly time consuming and its contains much more issues when operating under multiple devices and platforms therefore to avoid these issue we can combine multiple platforms to gather and achieve our ultimate target by using a microservices platform.* *Accordingly I think this project has perfectly illustrated and explained all major functionalities and operation”*

Prasad Jayasinghe  
Network Administrator – IT

*“Few pointers for future improvements would be to introduce common modules such as reporting, logs & service administration to better equip the services for practical world scenarios.”*

Chiranthana Weraniyagoda - BEng(Hons), MBA

|  |  |
| --- | --- |
| Evaluation module | Summary of feedback |
| Solution and Prototype | Overall comment for the solution and prototype evaluation is that the solution presented and the prototype addresses the problem and tries to give a solution. |
| Review on feedback | The current system was developed as a proof of concept that a dataflow could be transpiled into a microservice however due to time restrictions not all types of development concepts were captured and taken for consideration. This suggestion can be highlighted as a future enhancement for the current system. Other criticism that the current system doesn’t provide sufficient functionality was again due to the time restrictions faced during the project execution and this suggestion too can be taken up as a future enhancement. |

Table ‑ Feedback Evauation - Solution and Prototype

## Usability Accuracy and Performance

This section was already evaluated during the Non Functional Requirements Testing under the testing chapter.

## Limitations of the solution and future enhancements

*“Few pointers for future improvements would be to introduce common modules such as reporting, logs & service administration to better equip the services for practical world scenarios.”*

Chiranthana Weraniyagoda - BEng(Hons), MBA

Identified future enhancements are discussed in the conclusion chapter.

# Self Evaluation

This section will be focusing on self-evaluation of the above identified evaluation criteria along with an evaluation on the overall project.

## Self evaluation on identified evaluation criteria

|  |  |
| --- | --- |
| **Criteria** | **Self evaluation** |
| Overall concept | The project hypothesis was a valid and a timely one with a good business prospective and providing a solution to such a problem was an achievement. The concept could have further expanded to include testing, GUI creation and release versioning to create a more complete solution |
| Scope and depth of the project | Though microservices are a broad area with a substantial depth, the area and the depth covered during the Microservices Platform project is at a satisfactory level. |
| System design, architecture and implementation | The design approach applied along with design decisions and design goals and the system architecture are industry accepted standards and the design of the system was specially focusing of achieving the highest possible scalability. The design of the system can be taken up as a high point of the project.  The techniques and technologies used during the implementation of the Microservices Platform system are critically evaluated cutting edge techniques and technologies and can be satisfied with the implementation of the Microservices Platform system |
| Solution and Prototype | The solution put forward to solve the problem has addressed issues the microservices domain and the solution presents a novel approach in modeling a microservice. The prototype provides a more than satisfactory solution to the problem from the readers’ perspective but the prototype could have provided more user friendly functionalities for developers. Though those features were identified during the requirement elicitation process they were not implemented due to time constraints but it can be taken up that both the solution and the prototype are functioning as a proof of concept to the problem. |
| Usability, performance and accuracy of the prototype | The non functional requirements were identified ahead of the implementation and the literature review concentrated heavily in finding ways to solve the most troublesome development shortcomings in the microservices domain. Usability and performance of the Microservices Platform system were achieved to an exceptional level. |

Table ‑ Self Evaluation

## Reflection on the Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **FR No** | **Use case Mapping** | **Priority Level** | **Status** |
| FR1 | Developers should be able to use the dataflow tool to drag and drop components to develop the microservice | C | Implemented |
| FR2 | Developers should be able to use the microservice platform to deploy the microservice to different environments | C | Implemented |
| FR3 | The microservice platform should deploy the microservice into a Docker container | C | Implemented |
| FR4 | QA/QC should be able to use the platform to test the service before deploying | I | Not Implemented |
| FR5 | The microservice platform should take care of deploying microservices with adequate service versioning | I | Not Implemented |
| FR6 | The microservice platform should take care of Continuous Delivery when deploying the microservice | D | Not Implemented |
| FR7 | The microservice platform should posses a service registry to handle communication | C | Implemented |
| FR8 | The microservice platform should use Swagger to expose the metadata of the input and output of microservice | I | Implemented |
| FR9 | The service registry should be able to manage multiple environments for each microservice | I | Implemented |
| FR10 | Dev Ops should be able to enter configuration details of different environments for each microservice | I | Implemented |
| FR11 | The health of each microservice should be easily monitored | D | Not Implemented |
| FR12 | If the health of a microservice is in trouble an alarm should fire | D | Not Implemented |
| FR13 | The microservice platform should be able to deploy to multiple platforms simultaneously | D | Implemented |
| FR14 | The lifecycle of the microservice should be managed by the platform | D |  |
| FR15 | It should be able for a developer to discover another microservice and use as a component in his microservice using the service registry | C | Implemented |
| FR16 | The microservice platform should handle the Release management of the microservice | D | Not Implemented |
| FR17 | API tagging and taxonomy should be possible with the microservice platform to make it easier to find a microservice in the service registry | D | Not Implemented |

Table ‑ Reflection of Functional Requirements

Many obstacles had to be overcome during the course of the project and there were many successes and failures as well. To overcome the obstacles extensive level of research material were referred and feedbacks were obtained from experts on various fields and as a result project plan had to be revised several times to accommodate the extra time that was required to overcome the problems. But all in all in the researcher’s opinion, the project was successful to a great extent. When comparing the proposed system with current existing use cases researched during the Literature review, the newly designed system was researched, designed and developed specifically to solve the most troublesome development shortcomings in the microservices domain. The testing and domain expert evaluations have justified the novelty of the Microservices Platform system but there are a set of possible future enhancement that can be incorporated to the Microservices Platform system (refer to Conclusion Chapter) that would help to lessen the development load of microservices and increase developer throughput. Throughout the project many academic and technical concepts and soft skills were either learnt or enhanced (refer to Conclusion Chapter) that could be beneficial for future engagements.



# Chapter Summary

This chapter started with describing evaluation criteria, evaluation methodology and selected different types of evaluators to evaluate the different phases of the Microservices Platform project along with the justifications for those selections. Then Questionnaire and Interviews were selected as the evaluation approaches with justification followed by the evaluation feedbacks discussion and reviewing. According to the received feedback it was highlighted that the concept of the project was a timely one and the idea has a good potential towards the being successful and most of the evaluators valued the effort. When reviewing the feedback for the depth and the scope of the project it was highlighted that the depth and the scope is at an acceptable level for a postgraduate level research. Most of the evaluators praised the design of the Microservices Platform system along with the design goals and decision and the selection of technologies for the implementation was highly appreciated as well. The solution and the prototype evaluation too yielded positive remarks from the evaluator and the evaluators highlighted that the non functional requirements of the Microservices Platform system are also at an acceptable level. Several negative comments from the reviewers challenged with valid reasons and some were identified as potential future enhancements. The critical evaluation chapter concluded with a self evaluation covering all the aspects of the Microservices Platform project. Next chapter will be the conclusion chapter where it will discuss how successfully the project aim and objectives were achieved and moving on to justifying learning out comes, highlighting the challenges faced and future enhancement of the project and as well as wrapping up of the project.

# Conclusion

Contents

* Chapter Overview
* Achievement of Aim and Objectives
* Utilizing Knowledge of Course Modules
* Use of existing skills
* Leaning Outcomes
* Problems and Challenges Faced
* Limitations Of the Research
* Future Enhancements
* Contributing Remarks
* Concluding Remarks



# Chapter Overview

This chapter will focuses on concluding the project by highlighting the achievement of the aim and objectives, the problems and challenges faced during project life cycle, limitations of the project, identified future enhancements and closing remarks.

# Achievement of Aim and Objectives

**Aim**

*Design, develop and evaluate a framework that will help web developers build microservice projects using a dataflow programming language*

The aim was successfully achieved with in the allocated time period and the prototype was qualitatively and quantitatively evaluated through a self-critique process, domain experts and end users.

**Objectives**

|  |  |
| --- | --- |
| Objective 1 | Prepare Terms of Reference |
| Prepare the Terms of Reference which defines the aims and objectives, features of the prototype, background, resource requirements, project deliverables and activity schedule. The TOR is included as Chapter 1 of this documentation | |
| Objective 2 | Literature Survey |
| Carry Out an in-depth literature survey on the following subject areas   * Microservices: paradigm, philosophy, design approaches, existing products and implementation case studies * Dataflow languages: existing languages and frameworks   The literature review is included as chapter 2 of this documentation | |
| Objective 3 | Selection of a software development methodology |
| Select a software development methodology that would be most suitable to carry out the various phases of the project. For a detailed explanation refer to chapter 3 | |
| Objective 4 | Requirement gathering process |
| Carryout an in-depth user requirement gathering phase with:   * Developers who use microservice frameworks to develop applications * Architects who design applications around microservice frameworks * Project managers who manage teams that develop applications using microservice frameworks   This is included in Chapter 4 | |
| Objective 5 | Prepare a software requirement specification (SRS) |
| Using the information gathered through the literature review, end user questionnaire and observations, domain expert interviews and personal evaluations prepare the software requirements specification to document the functional and non-functional requirements of the proposed system. This is included in Chapter 4 | |
| Objective 6 | Selection of software and hardware resources |
| Select the most appropriate technologies, tools, APIs, libraries, platforms, algorithms and hardware requirements to implement the prototype. This is included in Chapter 5 | |
| Objective 7 | Prepare software design specification |
| Prepare the design specification for the prototype according to the analyzed requirements gathered from the requirement gathering phase. This is included in Chapter 5 | |
| Objective 8 | Develop the prototype |
| Develop the prototype using the most appropriate software and hardware resources to full fill the user requirements identified in the requirement specification. This is included in Chapter 6 | |
| Objective 9 | Testing of the prototype |
| Formulate a testing plan, prepare test cases and conduct an in-depth testing of the system to identify bugs and check whether the required functional and non-functional requirements of the users are achieved from the developed prototype. This is included in Chapter 7 | |
| Objective 10 | Evaluation of the work carried out |
| * Carry out a critical evaluation of the prototype using selected user groups of the system and conduct a review of the evaluation findings to determine how far the project has successfully addressed the hypothesis * Carry out a review with domain experts of different research areas used in the prototype to identify areas for future improvements * Perform a personal evaluation to self asses the work carried out   This is included in Chapter 8 | |
| Objective 11 | Documentation |
| Document all findings and key steps involved in the project and submit the project report. Chapters from 1 to 9 contain the documentation of the each objective of the project. | |

# Utilizing of Knowledge from Course Modules

Even though some modules of the M.Sc. program were not directly connected to the project research area, the knowledge and experience gathered through those contributed immensely toward the project success. The following modules distinguished themselves by providing significant contributions to the project completion.

|  |  |
| --- | --- |
| Module | Enterprise Development |
| Learning concepts of SOA and web services helped contribute greatly to this project | |
| Module | Research Methods and Professional Practice |
| Research methods module contributed to the initial project hypothesis identification and validation and later for the reviewing of various aspects related to the project. | |

# Use of existing skills

* Existing structured design skills such as context diagrams and flow charts were exploited successfully for the designing process of the system.
* Existing programming skills on Java Script language and NodeJS were utilized successfully for the completion of the project.

# Learning outcomes

* Though the course content offered during the M.Sc. program provided knowledge on various software engineering related topics they didn’t cover microservices in any form which was the basic foundation of the project. Therefore self-learning, online documentation and discussion with domain experts were used to gain the necessary knowledge to complete this project.
* Key technologies used in the project such as NodeJS and MongoDB were self-learnt through online documentation and tutorials for the successful completion of the project.
* Evaluating the project qualitatively and quantitatively required sound knowledge on software quality assurance and testing skills and these knowledge were enhance through self-learning process.
* Critical thinking and formal documentation skills were developed through gradual learning and hands on experience.

It can be concluded that knowledge gained through the M.Sc. curriculum, existing skills and self-learned skills combined resulted in the successful completion of the project.

# Problems and Challenges Faced

**The extensive scope**

Microservices though still in its infancy is heavily gaining traction in the software community and while still growing, is already a very broad and deep area. Though it was thought at the start of the project that the scope would be too small for a postgraduate level research with the in-depth research carried out it was evident that the scope would increase to a level which would not be able tackle within the allocated time period. Literature review and requirement elicitation processes were used to scale down the scope to a manageable level.

**Lack of academic publications**

While the domain itself was huge, papers discussing microservices itself as a subject was scarce. However a myriad of case study papers existed. Therefore the researcher was forced to turn the literature review into a numbers game.

Collecting as many case studies together as possible, each of the papers needed to be analyzed and the key concepts use extracted out. These concepts were all then tallied together and largely the most popular concepts and the ones that had the most promising results were then filtered out to be used in the design.

**Lack of knowledge and learning resources of key technologies**

Lack of knowledge with key technologies such as mongo DB and service registry concepts was a major problem face during the implementation of the prototype. Though there were online tutorials for those technologies only a handful was there describing the integrating of them together. This problem was overcome by detailed self-study and experimentations.

**Time Constraint**

Being a research project the inherited risk of frequent requirement fluctuations and lack of domain knowledge threatened to overrun the time allocated to the project. A possible solution to this problem was found by adapting the spiral development approach, which promises to keep the development iterative and support requirement fluctuations. It immensely helped to lower the time constraints and lead the project to a successful completion.

# Limitations of the Research

The lack of publications discussing microservices as a subject hindered much of the research early on, forcing the researcher of come up with a different approach to evaluating key technology concepts to implement in the design of the prototype.

# Future Enhancements

|  |  |  |  |
| --- | --- | --- | --- |
| **Enhancement ID** | ENH1 | **Priority Level** | High |
| **Enhancement** | **QA/QC should be able to use the platform to test the service before deploying** | | |
| **Description** | Software testing is a major point in the development lifecycle. While many case studies mention the difficulties of testing and the requirements gathering section touched on this heavily, due to time constraints this section was not implemented due to the scope of the feature rivaling that of the rest of the project combined. Future enhancers of the project can review the documentation of this project and consider developing various testing tools for several levels of testing the microservice being developed. | | |
| **Enhancement ID** | ENH2 | Priority Level | Medium |
| **Enhancement** | **The microservice platform should take care of deploying microservices with adequate service versioning** | | |
| **Description** | Service versioning helps keep track of changes between multiple releases of the same project and is now the most standardized way of being able to tell the difference between two versions. However due to time constraints this feature has been entirely neglected. Future enchanters can focus on this and build a robust versioning system that is directly integrated into the platform much like how Github has integrated versioning into its code management tool | | |
| **Enhancement ID** | ENH3 | **Priority Level** | Low |
| **Enhancement** | **The health of each microservice should be easily monitored** | | |
| **Description** | A health monitoring system is critical for microservices as should one microservice go down, all its depending microservices cease to function correctly. A possible approach to take would be to incorporate Nagios(Nagios Enterprises n.d.) instead of creating something from the ground up | | |
| **Enhancement ID** | ENH4 | **Priority Level** | High |
| **Enhancement** | **The microservice platform should handle the Release management of the microservice** | | |
| **Description** | Software lifecycle management is a critical part of a Manager’s job in managing a project’s development. An integrated lifecycle management system would ease the burden on a Manager by automating the process | | |
| **Enhancement ID** | ENH5 | **Priority Level** | Low |
| **Enhancement** | **API tagging and taxonomy should be possible with the microservice platform to make it easier to find a microservice in the service registry** | | |
| **Description** | While it is now easier to create microservice and integrate them together, as more and more developers start adapting the platform; the list of projects start getting larger and larger. Consequently it starts becoming harder and harder to find a microservice that fits your requirements. A taxonomy system tightly integrated into the platform will begin to ease the burdens of developers when looking for microservices to integrate with their project | | |
| **Enhancement ID** | ENH6 | **Priority Level** | Low |
| **Enhancement** | *“Few pointers for future improvements would be to introduce common modules such as reporting, logs & service administration to better equip the services for practical world scenarios.”*  Chiranthana Weraniyagoda - BEng(Hons), MBA | | |
| **Description** | A logging and reporting system would certainly be beneficial to the platform to make it more complete as a product | | |

Table ‑ Future Enhancements

# Contributing Remarks

The Microservice Platform aims to provide a solution to the current limitation faced by Microservice developers when developing their projects. While case studies exist that target only specific niches, this approach is the first of its kind to build a generalized microservice platform that all developers can use. The developed system has contributed to the domain by solving many of the issues developers face like, unwieldy IDE’s, huge codebases and inter service communication. It has also provided a one stop shop for all types of developers to come together and rapidly and easily develop microservice projects without being bogged down by the hassle of monolithic development practices.

# Concluding Remarks

The Dataflow based Microservice Platform provides a novel approach to the development of microservice by removing the burdens of monolithic development approaches and unwieldy IDEs from the developers and allowing developers to easily integrate microservices with each other without the need for extra developer overhead. The presented solution can be vastly beneficial to the software community, and since the solution itself is generalized; it can be integrated into any software product or domain allowing developers to build projects with the platform acting as a base foundation, removing much overhead from the entire team’s hands. This research also opens up many new domains of research, especially in the testing, lifecycle and version management areas of software; by allowing researchers to focus on the future enhancements mentioned above.

# Appendix A

|  |  |
| --- | --- |
| **Use case name** | Use Microservice |
| **Description** | This use case is to show the process where an external 3rd party consumes a deployed microservice |
| **Participating actors** | API Consumer |
| **Pre-conditions** | 1. System should be functional 2. There should be a deployed microservice |
| **Extended use cases** | None |
| **Included use cases** | None |
| **Main flow** | 1. API Consumer searches the service registry for relevant microservice 2. API Consumer enters the microservice location details into a REST client 3. API Consumer consumes the REST API |
| **Alternative flows** | None |
| **Exceptional flows** |  |
| **Post conditions** |  |

|  |  |
| --- | --- |
| **Use case name** | Sandbox Testing |
| **Description** | This use case is to show the process where an external 3rd party consumes a deployed microservice |
| **Participating actors** | API Consumer |
| **Pre-conditions** | 1. System should be functional 2. There should be a deployed microservice |
| **Extended use cases** | None |
| **Included use cases** | None |
| **Main flow** | 1. API Consumer searches the service registry for relevant microservice 2. API Consumer enters the microservice location details into a REST testing tool 3. API Consumer tests deployed microservice to check if it meets the specifications required to be used in required project |
| **Alternative flows** | None |
| **Exceptional flows** |  |
| **Post conditions** |  |

|  |  |
| --- | --- |
| **Use case name** | Configuration Management |
| **Description** | This use case is to show the process DevOps engineers and developers manage the platform by changing required configurations |
| **Participating actors** | Developers, DevOps Engineers |
| **Pre-conditions** | 1. System should be functional 2. There should be a deployed microservice |
| **Extended use cases** | None |
| **Included use cases** | None |
| **Main flow** | 1. Devops Engineer open the microservice platform 2. Devops Engineer goes to the configuration management section of the platform 3. DevOps engineer opens the section related to environments 4. DevOps Engineer adds new environment configuration 5. DevOps Engineer saves the configuration 6. Developer opens a microservice project 7. Devloper adds the environment configuration to his project |
| **Alternative flows** | None |
| **Exceptional flows** |  |
| **Post conditions** |  |

|  |  |
| --- | --- |
| **Use case name** | Monitor Container Health |
| **Description** | This use case is to show the process DevOps engineers use to monitor Container’s health |
| **Participating actors** | DevOps Engineers |
| **Pre-conditions** | 1. System should be functional 2. There should be a deployed microservice |
| **Extended use cases** | None |
| **Included use cases** | Monitor the platform, Fire Alarm |
| **Main flow** | 1. DevOps Engineer open the microservice platform 2. DevOps Engineer opens the section of teh tool that monitors the platform 3. DevOps Engineer monitors the health of each deployed microservice and its containers via series of charts and graphs |
| **Alternative flows** | None |
| **Exceptional flows** |  |
| **Post conditions** |  |

|  |  |
| --- | --- |
| **Use case name** | Fire Alarms |
| **Description** | This use case is to show the process where the platform fires an alarm if the platform’s health turns to the negative |
| **Participating actors** | DevOps Engineers |
| **Pre-conditions** | 1. System should be functional 2. There should be a deployed microservice 3. One or more container’s health must be declining 4. The overall platform’s health must be declining |
| **Extended use cases** | None |
| **Included use cases** | None |
| **Main flow** | 1. Microservice Platform detects container’s health is deteriorating 2. Microservice Platform fires off alarm to relevant systems |
| **Alternative flows** | None |
| **Exceptional flows** |  |
| **Post conditions** |  |

|  |  |
| --- | --- |
| **Use case name** | Deploy microservice to management platform |
| **Description** | This use case is to show the process where a microservice is deployed to the qa/staging server |
| **Participating actors** | Developers |
| **Pre-conditions** | 1. System should be functional 2. There should be a developed microservice |
| **Extended use cases** | None |
| **Included use cases** | Deploy to QA/ Staging server, Deploy to production environment |
| **Main flow** | 1. Developer opens a microservice project 2. Developer presses the deploy button 3. Developer selects which environments to deploy to 4. Developer deploys the microservice into a container |
| **Alternative flows** | None |
| **Exceptional flows** |  |
| **Post conditions** |  |

|  |  |
| --- | --- |
| **Use case name** | Lifecycle Management |
| **Description** | This use case is to show the process of managing the lifecycle of the project |
| **Participating actors** | Developer, Project Manager |
| **Pre-conditions** | 1. There should be a designed microservice |
| **Extended use cases** | None |
| **Included use cases** | Release management |
| **Main flow** | 1. Project manager decides whether to move the lifecycle of the project between development, testing, and production. 2. The developer sets the status of the project in the platform |
| **Alternative flows** | None |
| **Exceptional flows** |  |
| **Post conditions** |  |

|  |  |
| --- | --- |
| **Use case name** | Release Management |
| **Description** | This use case is to show the process of creating a release for the project |
| **Participating actors** | Project Manager |
| **Pre-conditions** | 1. There should be a developed microservice |
| **Extended use cases** | None |
| **Included use cases** | None |
| **Main flow** | 1. Project manager decides to updates the version of the microservice. 2. Developer opens the microservice project 3. The developer sets the status of the project in the platform |
| **Alternative flows** | None |
| **Exceptional flows** |  |
| **Post conditions** |  |

|  |  |
| --- | --- |
| **Use case name** | Version Management |
| **Description** | This use case is to show the process of creating versions of the microservice when updating the microservice |
| **Participating actors** | Project Manager |
| **Pre-conditions** | 1. There should be a deloyed microservice |
| **Extended use cases** | None |
| **Included use cases** | None |
| **Main flow** | 1. Project manager decides when to update the microservice with new features 2. Developer opens the microservice project 3. Developer creates new versions of the microservice and updates the features |
| **Alternative flows** | None |
| **Exceptional flows** |  |
| **Post conditions** |  |

|  |  |
| --- | --- |
| **Use case name** | API Tagging and taxonomy |
| **Description** | This use case is to show the process of tagging the microservice for better referencing in future projects |
| **Participating actors** | Architect, Tech Lead, Developer |
| **Pre-conditions** | 1. There should be a deployed microservice |
| **Extended use cases** | None |
| **Included use cases** | None |
| **Main flow** | 1. Architects and tech leads tag and taxonomize the project for future referencing 2. Developer opens the microservice project 3. Developer updates the tags in the project 4. Developer saves the project |
| **Alternative flows** | None |
| **Exceptional flows** |  |
| **Post conditions** |  |

# Appendix B

## Dataflow Tool Testing

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Case No | Description | Chrome | Firefox | Opera | Safari | IE 7 | IE 8 | IE 9 | IE 10 | IE 11 | EDGE | Pass Rate | Pass Rate without IE |
| 1 | Login to System | Pass | Pass | Pass | Pass | Fail | Fail | Fail | Fail | Fail | Pass | 50% | 100% |
| 2 | Add Project | Pass | Pass | Pass | Pass | Fail | Fail | Fail | Fail | Fail | Pass | 50% | 100% |
| 3 | Add Environment | Pass | Pass | Pass | Pass | Fail | Fail | Fail | Fail | Fail | Pass | 50% | 100% |
| 4 | Open Dataflow tool | Pass | Pass | Pass | Pass | Fail | Fail | Fail | Fail | Fail | Pass | 50% | 100% |

## Begin Component Testing

### DataTransferObject Testing

|  |  |  |
| --- | --- | --- |
| Case No | Description | Status |
| 1 | Creating an Integer | Pass |
| 2 | Creating a String | Pass |
| 3 | Creating a Double | Pass |
| 4 | Creating a Float | Pass |
| 5 | Creating a String Array | Pass |
| 6 | Creating an Integer Array | Pass |
| 7 | Creating a Double Array | Pass |
| 8 | Creating a Float Array | Pass |
| 9 | Creating a complex object with 1 String Parameter | Pass |
| 10 | Creating an Array of above Object | Pass |
| 11 | Creating a complex object with 1 Integer Parameter | Pass |
| 12 | Creating an Array of above Object | Pass |
| 13 | Creating a complex object with 1 Double Parameter | Pass |
| 14 | Creating an Array of above Object | Pass |
| 15 | Creating a complex object with 1 Float Parameter | Pass |
| 16 | Creating an Array of above Object | Pass |
| 17 | Creating an Array of Arrays | Pass |

### Input Object Testing

|  |  |  |
| --- | --- | --- |
| Case No | Description | Status |
| 1 | Creating an Integer | Pass |
| 2 | Creating a String | Pass |
| 3 | Creating a Double | Pass |
| 4 | Creating a Float | Pass |
| 5 | Creating a String Array | Pass |
| 6 | Creating an Integer Array | Pass |
| 7 | Creating a Double Array | Pass |
| 8 | Creating a Float Array | Pass |
| 9 | Creating a complex object with 1 String Parameter | Pass |
| 10 | Creating an Array of above Object | Pass |
| 11 | Creating a complex object with 1 Integer Parameter | Pass |
| 12 | Creating an Array of above Object | Pass |
| 13 | Creating a complex object with 1 Double Parameter | Pass |
| 14 | Creating an Array of above Object | Pass |
| 15 | Creating a complex object with 1 Float Parameter | Pass |
| 16 | Creating an Array of above Object | Pass |
| 17 | Creating an Array of Arrays | Pass |

## Database Query Component Testing

|  |  |  |
| --- | --- | --- |
| Case No | Description | Status |
| 1 | Select one parameter from database | Pass |
| 2 | Select two parameters from database | Pass |
| 3 | Select three parameters from database | Pass |
| 4 | Select using where clause with Integer | Pass |
| 5 | Select using where clause with String | Pass |
| 6 | Select using where clause with Double | Pass |
| 7 | Select using where clause with Float | Pass |

## Script Component Testing

|  |  |  |
| --- | --- | --- |
| Case No | Description | Status |
| 1 | Creating an Integer | Pass |
| 2 | Creating a String | Pass |
| 3 | Creating a Double | Pass |
| 4 | Creating a Float | Pass |
| 5 | Creating a String Array | Pass |
| 6 | Creating an Integer Array | Pass |
| 7 | Creating a Double Array | Pass |
| 8 | Creating a Float Array | Pass |
| 9 | Creating a complex object with 1 String Parameter | Pass |
| 10 | Creating an Array of above Object | Pass |
| 11 | Creating a complex object with 1 Integer Parameter | Pass |
| 12 | Creating an Array of above Object | Pass |
| 13 | Creating a complex object with 1 Double Parameter | Pass |
| 14 | Creating an Array of above Object | Pass |
| 15 | Creating a complex object with 1 Float Parameter | Pass |
| 16 | Creating an Array of above Object | Pass |
| 17 | Creating an Array of Arrays | Pass |
| 18 | Access Integer DataTransferObject | Pass |
| 19 | Access String DataTransferObject | Pass |
| 20 | Access Double DataTransferObject | Pass |
| 21 | Access Float DataTransferObject | Pass |
| 22 | Access Complex DataTransferObject | Pass |
| 23 | Access Array DataTransferObject | Pass |
| 24 | Test if condition | Pass |
| 25 | Test Switch Case | Pass |
| 26 | Test loops | Pass |

## Decision Component Testing

|  |  |  |
| --- | --- | --- |
| Case No | Description | Status |
| 1 | Access Integer DataTransferObject and check == condition with hardcoded variable | Pass |
| 2 | Access Integer DataTransferObject and check == condition with other DataTransferObject | Pass |
| 3 | Access String DataTransferObject and check == condition with hardcoded variable | Pass |
| 4 | Access String DataTransferObject and check == condition with other DataTransferObject` | Pass |
| 5 | Access Float DataTransferObject and check == condition with hardcoded variable | Pass |
| 6 | Access Float DataTransferObject and check == condition with other DataTransferObject | Pass |
| 7 | Access Double DataTransferObject and check == condition with hardcoded variable | Pass |
| 8 | Access Double DataTransferObject and check == condition with other DataTransferObject | Pass |
| 9 | Access Integer DataTransferObject and check != condition with hardcoded variable | Pass |
| 10 | Access Integer DataTransferObject and check != condition with other DataTransferObject | Pass |
| 11 | Access String DataTransferObject and check != condition with hardcoded variable | Pass |
| 12 | Access String DataTransferObject and check != condition with other DataTransferObject` | Pass |
| 13 | Access Float DataTransferObject and check != condition with hardcoded variable | Pass |
| 14 | Access Float DataTransferObject and check != condition with other DataTransferObject | Pass |
| 15 | Access Double DataTransferObject and check != condition with hardcoded variable | Pass |
| 16 | Access Double DataTransferObject and check != condition with other DataTransferObject | Pass |
| 17 | Access Integer DataTransferObject and check > condition with hardcoded variable | Pass |
| 18 | Access Integer DataTransferObject and check > condition with other DataTransferObject | Pass |
| 19 | Access String DataTransferObject and check > condition with hardcoded variable | Pass |
| 20 | Access String DataTransferObject and check > condition with other DataTransferObject` | Pass |
| 21 | Access Float DataTransferObject and check > condition with hardcoded variable | Pass |
| 22 | Access Float DataTransferObject and check > condition with other DataTransferObject | Pass |
| 23 | Access Double DataTransferObject and check > condition with hardcoded variable | Pass |
| 24 | Access Double DataTransferObject and check > condition with other DataTransferObject | Pass |
| 25 | Access Integer DataTransferObject and check >= condition with hardcoded variable | Pass |
| 26 | Access Integer DataTransferObject and check >= condition with other DataTransferObject | Pass |
| 27 | Access String DataTransferObject and check >= condition with hardcoded variable | Pass |
| 28 | Access String DataTransferObject and check >= condition with other DataTransferObject` | Pass |
| 29 | Access Float DataTransferObject and check >= condition with hardcoded variable | Pass |
| 30 | Access Float DataTransferObject and check >= condition with other DataTransferObject | Pass |
| 31 | Access Double DataTransferObject and check >= condition with hardcoded variable | Pass |
| 32 | Access Double DataTransferObject and check >= condition with other DataTransferObject | Pass |
| 33 | Access Integer DataTransferObject and check < condition with hardcoded variable | Pass |
| 34 | Access Integer DataTransferObject and check < condition with other DataTransferObject | Pass |
| 35 | Access String DataTransferObject and check < condition with hardcoded variable | Pass |
| 36 | Access String DataTransferObject and check < condition with other DataTransferObject` | Pass |
| 37 | Access Float DataTransferObject and check < condition with hardcoded variable | Pass |
| 38 | Access Float DataTransferObject and check < condition with other DataTransferObject | Pass |
| 39 | Access Double DataTransferObject and check < condition with hardcoded variable | Pass |
| 40 | Access Double DataTransferObject and check < condition with other DataTransferObject | Pass |
| 41 | Access Integer DataTransferObject and check <= condition with hardcoded variable | Pass |
| 42 | Access Integer DataTransferObject and check <= condition with other DataTransferObject | Pass |
| 43 | Access String DataTransferObject and check <= condition with hardcoded variable | Pass |
| 44 | Access String DataTransferObject and check <= condition with other DataTransferObject` | Pass |
| 45 | Access Float DataTransferObject and check <= condition with hardcoded variable | Pass |
| 46 | Access Float DataTransferObject and check <= condition with other DataTransferObject | Pass |
| 47 | Access Double DataTransferObject and check <= condition with hardcoded variable | Pass |
| 48 | Access Double DataTransferObject and check <= condition with other DataTransferObject | Pass |

## Database Insert Component Testing

|  |  |  |
| --- | --- | --- |
| Case No | Description | Status |
| 1 | Insert one parameter from database | Pass |
| 2 | Insert two parameters from database | Pass |
| 3 | Insert three parameters from database | Pass |
| 4 | Update using where clause with Integer | Pass |
| 5 | Update using where clause with String | Pass |
| 6 | Update using where clause with Double | Pass |
| 7 | Update using where clause with Float | Pass |

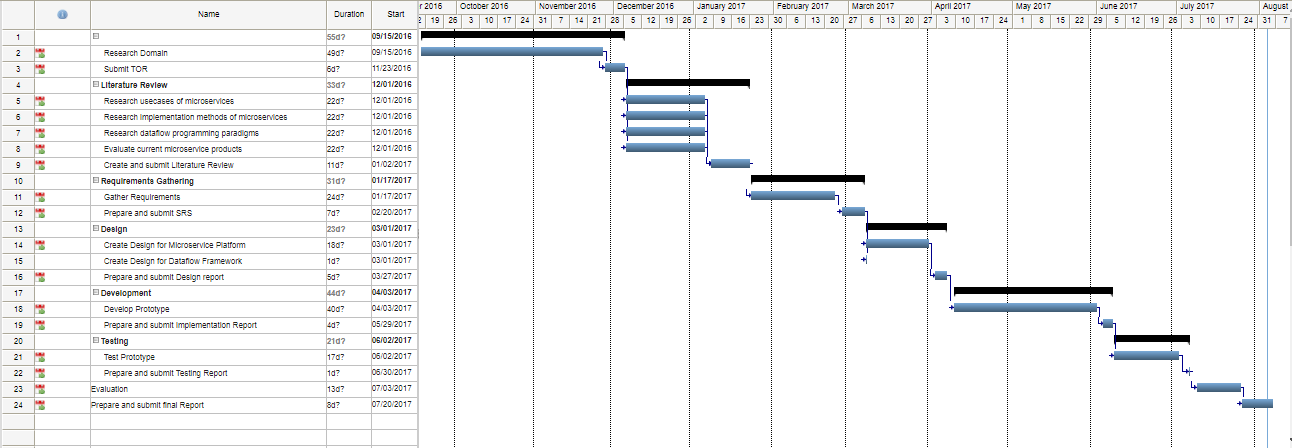
## REST Component Testing

|  |  |  |
| --- | --- | --- |
| Case No | Description | Status |
| 1 | GET testing with DataTransferObjects | Pass |
| 2 | POST testing with DataTransferObjects | Pass |
| 3 | PUT testing with DataTransferObjects | Pass |
| 4 | DELETE testing with DataTransferObjects | Pass |

## End Component Testing

|  |  |  |
| --- | --- | --- |
| Case No | Description | Status |
| 1 | Creating an Integer | Pass |
| 2 | Creating a String | Pass |
| 3 | Creating a Double | Pass |
| 4 | Creating a Float | Pass |
| 5 | Creating a String Array | Pass |
| 6 | Creating an Integer Array | Pass |
| 7 | Creating a Double Array | Pass |
| 8 | Creating a Float Array | Pass |
| 9 | Creating a complex object with 1 String Parameter | Pass |
| 10 | Creating an Array of above Object | Pass |
| 11 | Creating a complex object with 1 Integer Parameter | Pass |
| 12 | Creating an Array of above Object | Pass |
| 13 | Creating a complex object with 1 Double Parameter | Pass |
| 14 | Creating an Array of above Object | Pass |
| 15 | Creating a complex object with 1 Float Parameter | Pass |
| 16 | Creating an Array of above Object | Pass |
| 17 | Creating an Array of Arrays | Pass |

# Appendix C



# References

Abbasi, M.Q. et al., 2012. Modeling and Evaluating User Interface Aesthetics Employing ISO 25010 Quality Standard. In *2012 Eighth International Conference on the Quality of Information and Communications Technology*. IEEE, pp. 303–306. Available at: http://ieeexplore.ieee.org/document/6511832/ [Accessed January 22, 2017].

Abbott, M.L. & Fisher, M.T., 2009. *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise*, Pearson Education. Available at: https://books.google.lk/books?id=yzUpD2YbhWwC [Accessed December 28, 2016].

Adobe, Brackets - A modern, open source code editor that understands web design. Available at: http://brackets.io/ [Accessed July 16, 2017].

Alavi, M., 1984. An Assessment of the Prototyping Approach To Information Systems Development. *Communications of the ACM*, 27(6), pp.556–563.

Amaral, M. et al., 2015. Performance Evaluation of Microservices Architectures Using Containers. In *2015 IEEE 14th International Symposium on Network Computing and Applications*. IEEE, pp. 27–34. Available at: http://ieeexplore.ieee.org/document/7371699/ [Accessed January 2, 2017].

Amazon Inc, Amazon Simple Storage Service (S3) - Cloud Storage. Available at: https://aws.amazon.com/s3/?nc2=h\_m1 [Accessed January 11, 2017a].

Amazon Inc, Amazon Web Services (AWS) - Cloud Computing Services. Available at: https://aws.amazon.com/ [Accessed January 11, 2017b].

Amazon Inc, AWS Elastic Beanstalk – Deploy Web Applications. Available at: https://aws.amazon.com/elasticbeanstalk/?nc2=h\_m1 [Accessed January 11, 2017c].

Amazon Inc, AWS Lambda - Serverless Compute. Available at: https://aws.amazon.com/lambda/?nc2=h\_m1 [Accessed January 11, 2017d].

Amazon Inc, Docker Basics - Amazon EC2 Container Service. Available at: http://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html [Accessed January 11, 2017e].

Amazon Inc, Elastic Compute Cloud (EC2) Cloud Server & Hosting – AWS. Available at: https://aws.amazon.com/ec2/ [Accessed January 11, 2017f].

Baker, T., 2013. Improve Your Software Architecture with Ports and Adapters. Available at: https://spin.atomicobject.com/2013/02/23/ports-adapters-software-architecture/ [Accessed January 26, 2017].

Bassil, Y., 2012. A Simulation Model for the Waterfall Software Development Life Cycle. Available at: http://arxiv.org/abs/1205.6904 [Accessed July 30, 2017].

Boehm, B.W., 1988. Spiral Model of Software Development and Enhancement. *Computer*, 21(5), pp.61–72.

BPMN, BPMN Specification - Business Process Model and Notation. Available at: http://www.bpmn.org/ [Accessed January 22, 2017a].

BPMN, Web-based tooling for BPMN, DMN and CMMN | bpmn.io. Available at: https://bpmn.io/ [Accessed January 22, 2017b].

Bpmn-io, Example BPMN Diagram. Available at: https://github.com/bpmn-io/bpmn-js-examples/tree/master/simple-bower.

British Standards Institute, 1998. BS 7925-2 Software component testing.

Brown, P.C., 2008. *Implementing SOA: Total Architecture in Practice*, Pearson Education. Available at: https://books.google.lk/books?id=3EfZKxJ7OFkC [Accessed December 28, 2016].

C9, C9. Available at: https://c9.io/sajithdil [Accessed July 16, 2017].

Casterson, S., 2016. *Microservices: A Guide for Beginners*, Lulu.com. Available at: https://books.google.lk/books?id=9KmhCwAAQBAJ [Accessed December 28, 2016].

Crockford, D., Douglas Crockford: The JavaScript Programming Language - YouTube. Available at: https://www.youtube.com/watch?v=v2ifWcnQs6M&list=PL62E185BB8577B63D [Accessed July 16, 2017].

Docker Inc, Docker - Build, Ship, and Run Any App, Anywhere. Available at: https://www.docker.com/ [Accessed January 11, 2017a].

Docker Inc, Docker Engine \textbar Docker. Available at: https://www.docker.com/products/docker-engine [Accessed January 11, 2017b].

Docker Inc, Docker Remote API - Docker. Available at: https://docs.docker.com/engine/reference/api/docker\_remote\_api/ [Accessed January 11, 2017c].

Douglas, B., Service-Oriented Architecture (SOA) Definition. Available at: http://www.service-architecture.com/articles/web-services/service-oriented\_architecture\_soa\_definition.html [Accessed December 28, 2016].

Dubois, A. & Gadde, L.-E., 2002. Systematic combining: an abductive approach to case research. *Journal of Business Research*, 55(7), pp.553–560. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0148296300001958 [Accessed July 30, 2017].

Eclipse, Eclipse desktop &amp; web IDEs. Available at: https://eclipse.org/ide/ [Accessed July 16, 2017].

ECMA, JSON. Available at: http://www.json.org/ [Accessed January 12, 2017].

Erl, T., 2005. *Service-oriented architecture : concepts, technology, and design*, Upper Saddle River, N.J. [etc.]: Prentice Hall Professional Technical Reference.

Evans, C.C., The Official YAML Web Site. Available at: http://yaml.org/ [Accessed January 12, 2017].

Fetzer, C., 2016. Building Critical Applications Using Microservices. *IEEE Security & Privacy*, 14(6), pp.86–89. Available at: http://ieeexplore.ieee.org/document/7782696/ [Accessed January 21, 2017].

Gadea, C. et al., 2016. A microservices architecture for collaborative document editing enhanced with face recognition. In *2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. IEEE, pp. 441–446. Available at: http://ieeexplore.ieee.org/document/7507409/ [Accessed January 21, 2017].

Google Inc, AngularJS: Developer Guide: Internet Explorer Compatibility. Available at: https://docs.angularjs.org/guide/ie [Accessed July 24, 2017a].

Google Inc, AngularJS — Superheroic JavaScript MVW Framework. Available at: https://angularjs.org/ [Accessed July 16, 2017b].

Google Inc, gcloud docker \textbar Cloud SDK \textbar Google Cloud Platform. Available at: https://cloud.google.com/sdk/gcloud/reference/docker [Accessed January 11, 2017c].

Google Inc, Google Cloud Platform. Available at: https://console.cloud.google.com/appengine?src=ac&pli=1 [Accessed January 11, 2017d].

Google Inc, Microservices Architecture on Google App Engine \textbar App Engine standard environment for Python \textbar Google Cloud Platform. Available at: https://cloud.google.com/appengine/docs/python/microservices-on-app-engine [Accessed January 11, 2017e].

Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad; Buckley, A., 2014. The Java® Language Specification - jls8.pdf. *Addison-Wesley*, p.688. Available at: https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf.

Heorhiadi, V. et al., 2016. Gremlin: Systematic Resilience Testing of Microservices. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, pp. 57–66. Available at: http://ieeexplore.ieee.org/document/7536505/ [Accessed January 2, 2017].

Humble, J., What is Continuous Delivery? - Continuous Delivery. Available at: https://continuousdelivery.com/ [Accessed January 26, 2017].

Inc, F., React - A JavaScript library for building user interfaces. Available at: https://facebook.github.io/react/ [Accessed July 16, 2017].

jgraph, mxGraph. Available at: https://github.com/jgraph/mxgraph.

Johnston, W.M., Hanna, J.R.P. & Millar, R.J., 2004. Advances in dataflow programming languages. *ACM Comput. Surv.*, 36(1), pp.1–34. Available at: http://portal.acm.org/citation.cfm?doid=1013208.1013209 [Accessed December 28, 2016].

Kent, A., 2000. *Encyclopedia of Library and Information Science: Volume 66 - Supplement 29 - Automated System for the Generation of Document Indexes to Volume Visualization*, Taylor & Francis. Available at: https://books.google.lk/books?id=FiiLbNN1q4MC [Accessed December 28, 2016].

Komodo, Komodo IDE: The Best IDE for Web and Mobile App Development | ActiveState. Available at: https://www.activestate.com/komodo-ide [Accessed July 16, 2017].

Krutz, D.E. & Lutz, M., 2013. Bug of the Day: Reinforcing the importance of testing. In *2013 IEEE Frontiers in Education Conference (FIE)*. IEEE, pp. 1795–1799. Available at: http://ieeexplore.ieee.org/document/6685147/ [Accessed January 22, 2017].

Le, D.M., Dang, D.-H. & Nguyen, V.-H., 2016. Domain-driven design patterns: A metadata-based approach. In *2016 IEEE RIVF International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*. IEEE, pp. 247–252. Available at: http://ieeexplore.ieee.org/document/7800302/ [Accessed January 26, 2017].

Linthicum, D.S., 2016. Practical Use of Microservices in Moving Workloads to the Cloud. *IEEE Cloud Computing*, 3(5), pp.6–9. Available at: http://ieeexplore.ieee.org/document/7742277/ [Accessed January 21, 2017].

Linux Foundation, Node.js. Available at: https://nodejs.org/en/ [Accessed July 16, 2017a].

Linux Foundation, Open API Initiative. Available at: https://www.openapis.org/ [Accessed January 12, 2017b].

Marston, T., What is the 3-Tier Architecture? Available at: http://www.tonymarston.net/php-mysql/3-tier-architecture.html [Accessed May 27, 2017].

Martin, J. & James, 1991. *Rapid application development*, Macmillan Pub. Co. Available at: http://dl.acm.org/citation.cfm?id=103275&CFID=791494732&CFTOKEN=96166979 [Accessed July 30, 2017].

Microsoft, Azure Service Fabric\textemdashBuilding Microservices \textbar Microsoft Azure. Available at: https://azure.microsoft.com/en-us/services/service-fabric/ [Accessed January 11, 2017a].

Microsoft, Visual Studio IDE, Code Editor, Team Services, &amp; Mobile Center. Available at: https://www.visualstudio.com/ [Accessed July 16, 2017b].

Microsoft, VPL Introduction. Available at: https://msdn.microsoft.com/en-us/library/bb483088.aspx [Accessed January 11, 2017c].

Microsoft, VPL Introduction. Available at: https://msdn.microsoft.com/en-us/library/bb483088.aspx [Accessed January 22, 2017d].

MongoDB inc, MongoDB for GIANT Ideas | MongoDB. Available at: https://www.mongodb.com/ [Accessed July 28, 2017].

Nagios Enterprises, Nagios - The Industry Standard In IT Infrastructure Monitoring. Available at: https://www.nagios.org/ [Accessed July 26, 2017].

Netbeans, Welcome to NetBeans. Available at: https://netbeans.org/ [Accessed July 16, 2017].

Newman, S., 2015. *Building Microservices: Designing Fine-Grained Systems*, O’Reilly Media. Available at: https://books.google.lk/books?id=RDl4BgAAQBAJ [Accessed December 28, 2016].

Nextflow, Nextflow - A DSL for parallel and scalable computational pipelines. Available at: https://www.nextflow.io/ [Accessed January 11, 2017].

Notepad++, Notepad++ Home. Available at: https://notepad-plus-plus.org/ [Accessed July 16, 2017].

of Technology. Automatic Programming Information Centre, 1972. *A.P.I.C. Studies in Data Processing*, Academic Press. Available at: https://books.google.lk/books?id=uXksAAAAIAAJ [Accessed December 28, 2016].

Openwire, Mitov Software - OpenWire. Available at: http://www.mitov.com/products/openwire#overview [Accessed January 11, 2017].

Peyrott, S., An Introduction to Microservices, Part 3: The Service Registry. Available at: https://auth0.com/blog/an-introduction-to-microservices-part-3-the-service-registry/ [Accessed January 26, 2017].

Piska, K., 2105. Working with legacy architecture · allegro.tech. Available at: http://allegro.tech/2015/01/working-with-legacy-architecture.html [Accessed January 26, 2017].

Preston-Werner, T., 2013. Semantic Versioning 2.0.0 - Semantic Versioning. Available at: http://semver.org/ [Accessed March 19, 2017].

Rahman, M. & Gao, J., 2015. A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development. In *2015 IEEE Symposium on Service-Oriented System Engineering*. IEEE, pp. 321–325. Available at: http://ieeexplore.ieee.org/document/7133548/ [Accessed January 2, 2017].

RAML Workgroup, RAML \textbar. Available at: http://raml.org/ [Accessed January 12, 2017].

Richardson, C., Monolithic Architecture pattern. *Microservice architecture*. Available at: http://microservices.io/patterns/monolithic.html [Accessed December 28, 2016].

Sill, A., 2016. The Design and Architecture of Microservices. *IEEE Cloud Computing*, 3(5), pp.76–80. Available at: http://ieeexplore.ieee.org/document/7742259/ [Accessed January 21, 2017].

SmartBear Software, Swagger – The World’s Most Popular Framework for APIs. Available at: http://swagger.io/ [Accessed January 12, 2017].

Sousa, G., Rudametkin, W. & Duchien, L., 2016. Automated Setup of Multi-Cloud Environments for Microservices-Based Applications. *9th IEEE International Conference on Cloud Computing*.

Spring, 3. Programming Model. Available at: http://docs.spring.io/spring-cloud-stream/docs/Brooklyn.RELEASE/reference/html/\_programming\_model.html [Accessed January 11, 2017].

Stranghöner, R., 2015. Self-Contained Systems. Available at: https://www.innoq.com/en/links/self-contained-systems-infodeck/ [Accessed January 26, 2017].

Stroppa, D., 2016. Run Containerized Microservices with Amazon EC2 Container Service and Application Load Balancer \textbar AWS Compute Blog. Available at: https://aws.amazon.com/blogs/compute/microservice-delivery-with-amazon-ecs-and-application-load-balancers/ [Accessed January 11, 2017].

Stubbs, J., Moreira, W. & Dooley, R., 2015. Distributed Systems of Microservices Using Docker and Serfnode. In *2015 7th International Workshop on Science Gateways*. IEEE, pp. 34–39. Available at: http://ieeexplore.ieee.org/document/7217926/ [Accessed January 12, 2017].

Taylor, P.S. et al., 2009. Agile Software Development. *Eth Mtec*, 1(6), pp.1–5.

Ueda, T., Nakaike, T. & Ohara, M., 2016. Workload characterization for microservices. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, pp. 1–10. Available at: http://ieeexplore.ieee.org/document/7581269/ [Accessed January 21, 2017].

Villamizar, M. et al., 2016. Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, pp. 179–182. Available at: http://ieeexplore.ieee.org/document/7515686/ [Accessed January 2, 2017].

W3, Extensible Markup Language (XML). Available at: https://www.w3.org/XML/ [Accessed January 12, 2017].

w3schools, Browser Statistics. Available at: https://www.w3schools.com/Browsers/default.asp [Accessed July 24, 2017].

Whatis, What is virtual machine (VM)? - Definition from WhatIs.com. Available at: http://searchservervirtualization.techtarget.com/definition/virtual-machine [Accessed January 11, 2017].

Williams, I.D., 2010. JavaScript Code Standards Conventions and Standards.

WSO2, WSO2 - Increase your pace of innovation. Available at: http://wso2.com/ [Accessed January 11, 2017a].

WSO2, WSO2 Microservices Framework for Java \textbar WSO2 Inc. Available at: http://wso2.com/products/microservices-framework-for-java/ [Accessed January 11, 2017b].

Zimmermann, G., 2007. Open user interface standards - towards coherent, task-oriented and scalable user interfaces in home environments. *Intelligent Environments, 2007. IE 07. 3rd IET International Conference on*, pp.36–39.