MicroDG

Distributed Data Retrieval & Generation through Microservices

David Brady
BSc. Hons in Computing 2015/2016
Specialisation: Cloud Computing
National College of Ireland
Student Number: x12112267

Email: x12112267@student.ncirl.ie

National College of Ireland

Project Submission Sheet – 2015/2016

School of Computing

Student Name:	David Brady		
Student ID:	X12112267		
Programme:	BSHC	**Year:**	4
Module:	Software Project		
Lecturer	Eamon Nolan		
Submission Due Date:	11/05/2016		
Project Title:	MicroDG – Distributed Data Retrieval & Generation through Mircoservices		
Word Count:	14,310 (excluding Appendix) 20,712 (including Appendix)		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: *David Brady*……..

Date: 09/05/2016…….

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. You must ensure that you retain a HARD COPY of ALL projects, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
3 Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Declaration Cover Sheet for Project Submission

SECTION 1 *Student to complete*

Name:	**David Brady**
Student ID:	**x12112267**
Supervisor:	**Manuel Tova Izquierdo**

SECTION 2 Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read the College statement on plagiarism (summarised overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Signature: *David Brady*_____ Date: 09/05/2016

NB. If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the College's Disciplinary Committee. Should the Committee be satisfied that plagiarism has occurred this is likely to lead to your failing the module and possibly to your being suspended or expelled from college.

What constitutes plagiarism or cheating?

The following is extracted from the college's formal statement on plagiarism as quoted in the Student Handbook. References to "assignments" should be taken to include any piece of work submitted for assessment.

Paraphrasing refers to taking the ideas, words or work of another, putting it into your own words and crediting the source. This is acceptable academic practice provided you ensure that credit is given to the author. Plagiarism refers to copying the ideas and work of another and misrepresenting it as your own. This is completely unacceptable and is prohibited in all academic institutions. It is a serious offence and may result in a fail grade and/or disciplinary action. All sources that you use in your writing must be acknowledged and included in the reference or bibliography section. If a particular piece of writing proves difficult to paraphrase, or you want to include it in its original form, it must be enclosed in quotation marks and credit given to the author.

When referring to the work of another author within the text of your project, you must give the author's surname and the date the work was published. Full details for each source must then be given in the bibliography at the end of the project.

Penalties for Plagiarism

If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the college's Disciplinary Committee. Where the Disciplinary Committee makes a finding that there has been plagiarism, the Disciplinary Committee may recommend:

- that a student's marks shall be reduced
- that the student be deemed not to have passed the assignment
- that other forms of assessment undertaken in that academic year by the same student be declared void
- that other examinations sat by the same student at the same sitting be declared void

Further penalties are also possible including:

- suspending a student from college for a specified time,
- expelling a student from college,
- prohibiting a student from sitting any examination or assessment,
- the imposition of a fine, and
- the requirement that a student attend additional or other lectures or courses or undertake additional academic work.

DOCUMENT CONTROL

This technical report is a live document and will be updated periodically throughout the lifecycle of the project. Updates to this document will be recorded in the table below:

Revision History

Date	Version	Action	Contributor
06/11/2015	1	Created	David Brady
30/01/2016	2	Modified to reflect new project concept	David Brady
04/02/2016	3	Submitted for mid-point evaluation	David Brady
23/04/2016	4	Continued from the mid-point version	David Brady

Distribution List

Name	Title/Role
Eamon Nolan	Lecturer
Manuel Tova Izquierdo	Academic Supervisor
Emma O'Brien	Client/Project Sponsor

ABSTRACT

The focus of the MicroDG project was to build a cloud-based system of microservices for the purpose of dynamically generating sales lead data. The system addresses the need for the automated bulk tagging of freely available business data with a variety of additional information such as geographical information. This type of task would normally be done manually which is neither time efficient nor financially viable for many businesses.

This project is also a study into the advantages of applying a microservice architecture pattern in partnership with a variety of programming languages and toolkits, in terms of system speed and scalability.

The system is currently in production using Scala, Akka, Spray and AngularJS amongst other technologies. A working prototype of this system is available for download and can be run in the local environment (please see Appendix E: User Manual, for further details).

Acknowledgements

Foremost, I would like to express my sincere gratitude to my academic supervisor Dr Manuel Tova Izquierdo. You have shown me a huge level of support and guidance throughout my final year and your interest in my work has been greatly appreciated.

To my family, thank you for always listening to me and for putting up with me over the last four years.

To my mother, thank you for always being there for me, keeping me well fed and helping me through a difficult year.

To my dear friends, Paul and Steve, your unwavering support and encouragement has given me strength and has kept me focused throughout my degree, thank you.

Last but in no way least, I want to say a special thank you to Emma. Your love and belief in me guided me through this mammoth task and always drove me to achieve my best. This thesis, this degree, these last four years, none of it would have been possible without you and for that I will be forever grateful.

TABLE OF CONTENTS

Document Control 5
Revision History 5
Distribution List 5
Abstract 6
Introduction 12
Aims 12
Background 13
Technologies 14
Development 14
Testing 15
Deployment 16
System 17
Requirements Specification 17
Requirements Categorisation 17
Functional Requirements 18
System Level Requirements 18
Service Level Functional Requirements 22
Non-Functional Requirements 31
System Level Non-Functional Requirements 31
Performance and Availability Requirements 31
Security Requirements 31
User Requirements 31
End User (EU) 31
Design and Architecture 32
System Architecture Diagram 32
System Use Case Model 33
Service Overview 34
Advantages of the Microservices Architecture 34
Characteristics of the Microservices Architecture 34
Implementation 36
Introduction to the MicroDG microservices 36

Common Feature: Spray HTTP Integration Layer 36
Common Feature: Cross Origin Resource Sharing 38
Microservice: Base Business Details Service 39
Service Overview 39
Composite Technologies 39
Code Base Overview 39
API Endpoints 40
Sample Response 42
Process Flow Description 43
Process Flow Diagram 43
Function/Method Highlight 44
Client/Server Sequence Diagram 45
Code Coverage 45
Microservice: Geo Tagging Service 46
Service Overview 46
Composite Technologies 46
Code Base Overview 46
API Endpoints 47
Sample Response 48
Process Flow Description 49
Process Flow Diagram 49
Function/Method Highlight 50
Client/Server Sequence Diagram 52
Code Coverage 52
Microservice: AWS Storage Service 53
Service Overview 53
Composite Technologies 53
Code Base Overview 53
API Endpoints 54
Process Flow Description 55
Process Flow Diagram 55
Function/Method Highlight 56

Client/Server Sequence Diagram 57
Microservice: Process Controller Service 58
Service Overview 58
Composite Technologies 58
Code Base Overview 58
API Endpoints 59
Sample Response 61
Process Flow Description 62
Function/Method Highlight 65
Client/Server Sequence Diagram 68
Code Coverage 68
Graphical User Interface (GUI) 69
Source Code Repository 69
Composite Technologies 69
Service Gateway 69
Landing/Home Page 70
Interface: Base Business Details Service 71
Interface: Geo Tagging Service 73
Interface: Process Controller Service 74
Interface: Analytics Dashboard 75
Library Dependencies 76
Testing 77
Unit Testing (White Box) 77
Overview 77
Test Frameworks 77
Running Tests 77
Base Business Details Service 78
Geo Tagging Service 79
Process Controller Service 80
End to End Testing (Black Box) 81
Overview 81
Test Frameworks 81

Running Tests 81
Base Business Details Service 82
Geo Tagging Service 83
Process Controller Service 84
User Acceptance Testing 85
Overview 85
Scenarios 85
User Experience 85
Group One: Project Sponsor 86
Group Two: Assorted User Group 87
Deployment 87
Deployment Strategy 88
Docker Deployment Overview 89
Deployment Diagram 89
Conclusions 90
Further Developments 91
Bibliography 92
Appendix 93
Appendix A: Project Proposal 93
Appendix B: Project Plan 97
Appendix C: Requirements Elicitation Transcription 99
Appendix D: Monthly Journals 100
Appendix E: User Manual 108
Cloning the source code 108
Compiling the services 108
Testing the services (Unit Tests) 108
Running the services 108
Starting the Client UI 109
Testing the UI (End-to-End Tests) 109
Interacting with the services 109
Appendix F: UAT Feedback Forms 110

INTRODUCTION

The modern world has taken to the cloud. We rely on online services to bring us everything from the food on our tables to the clothes on our back. When we want to watch TV shows or listen to music on demand, we go online. When we want to speak to or play a game with our friends, we go online.

In order for any new or existing business to succeed in this rapidly growing digital market, they must develop an online presence and harness the power of the cloud. Never before have we had such an opportunity for innovative businesses and entrepreneurs to create start-ups and deliver products with virtually no IT-related capital expenditure and this is all thanks to cloud computing.

Still, though, there is a problem and that problem is speed. We have the processing power at our fingertips and we can scale out our online applications at the click of a button. However, at a time when all businesses are trying to compete online, the currency is speed and too many applications cannot keep up.

An online article on GigaSpaces.com concerning latency stated that "Amazon found every 100ms of latency cost them 1% in sales. Google found an extra .5 seconds in search page generation time dropped traffic by 20%. A broker could lose $4 million in revenues per millisecond if their electronic trading platform is five milliseconds behind the competition." [1]

When it comes to the speed of a cloud-based application, there are certain factors that are out of the control of the software developer. These factors include the speed of the network as well as the service provider's hardware, amongst others. However, one factor which is very much in the hands of the developer is the architecture and technology used to develop these applications. With speed being of such a high priority to online businesses, the time has come to move away from monolithic applications which are tightly coupled, difficult to scale efficiently and restricted in terms of development languages and move towards a loosely coupled microservices architecture which focuses its priority on speed and scalability whilst using the best tools for the job.

Aims

The aim of this system is to develop a web-based application using Scala, Akka and Spray which demonstrates the advantages of both these technologies in unison and the microservices architecture pattern in terms of speed, scalability, modularity and concurrency.

When complete, the system will be deployed on the cloud and will be accessible to anyone via a web browser. All services will be accessible independently and as a logical composition.

The system will specifically address the requirements of the third party client and greatly improve the operational time for their internal field sales lead generation tasks.

[1] Liddle, Jim. "Amazon Found Every 100Ms Of Latency Cost Them 1% In Sales. Gigaspaces XAP Blog – The In-Memory Computing Platform". *Blog.gigaspaces.com*. N.p., 2016. Web. 4 Feb. 2016.

Background

I have three key motivations for building this proposed system:

1. I have spoken to Emma O'Brien, a key member of the Marketing team at Mcor. One of her daily tasks is compiling information for field sales agents. The typical process includes gathering data from their internal lead generation process and then manually gathering supporting data, for example geo coordinates for each lead's address. This task can be time-consuming based on the high volumes of leads that are generated on a weekly basis. The proposed MicroDG system was of interest to Ms O'Brien because she felt it could potentially reduce the time required to complete the above task. Ms O'Brien has offered to assist in the requirements elicitation process which will help to guide the final system design.
2. This system will allow me to demonstrate some of the skills I gained from my final year specification. Each service can be considered to be SAAS[2] and I will be utilising certain PAAS[3] offerings such as persistent storage as a service. Also, as some services (such as the Geo Tagging Service) will experience higher levels of activity, it will allow me to practise scalability strategies for individual services rather than on the system as a whole.
3. What is most important to me about this proposed system is not the concept behind the application but the technology stack I propose using to deliver it. I am very fortunate to have signed a permanent contract with a fantastic company, Workday, Inc., as an Associate Software Development Engineer, following a nine-month work placement I completed during the third year of my studies. The role I will be going into this coming June is one which uses some, if not all, of my proposed technology stack for this project. Also, by the time I return to Workday, my team will be in the process of migrating the product I worked on to a microservices architecture, so any experience gained now will be of a huge advantage to me. The concept of my final year project has no relevance to the product offerings of Workday but I see this project as an opportunity to become better versed in the tools and technology which I will be using in my future career.

[2] SAAS – Software as a Service
[3] PAAS – Platform as a Service

Technologies

Below is a list of the technologies I have used in the development, testing and deployment of this microservices system.

DEVELOPMENT

Scala

Scala, meaning “Scalable Language”, is concise and powerful type-safe Java Virtual Machine (JVM) language which incorporates both Object-Oriented and Functional programming together. Scala source code compiles to Java Bytecode and as a result runs on the JVM.

Scala is seen as a faster, less verbose and more powerful alternative to Java. However, Scala is designed to interoperate with Java and therefore can avail of Java classes and libraries.

I chose Scala as the primary development language for this system because of its use of actors which assist in building fast, scalable concurrent systems. Scala’s use of futures also allows for better programming of asynchronous services.

Spray

Spray is a lightweight toolkit/library used for building REST/HTTP integration layers on top of Scala and Akka applications.

The main component of Spray used in the system is the lightweight http server called “spray-can”. This http server allowed me to define a series of routes which handle incoming HTTP requests. The service(s) then utilises Akka’s actor system to manage concurrent threads.

Akka

Akka is a toolkit and runtime for building concurrent, distributed and resilient message-driven applications on the JVM.

Akka was used along with the spray-can http server to provide a concurrent REST/HTTP integration layer for each service.

SBT

SBT (Scala Build Tool) is a build tool for Scala. Like Maven, SBT handles build tasks, dependency management and deployment tasks.

AngularJS

AngularJS is a JavaScript framework which is integrated into HTML in order to produce powerful client side applications.

AngularJS was used to build highly functional and dynamic User Interfaces/Dashboards for the retrieval and display of data from the system's microservices.

Bootstrap, HTML5, CSS3 and JavaScript

The standard HTML5 mark-up language along with the Twitter Bootstrap libraries (CSS and JavaScript) were used to develop responsive cross platform/device web pages to allow the user to interact with the system's constituent microservices.

AWS S3

Amazon Web Services S3 (Simple Storage Service) offers simple but efficient object storage on the cloud.

AWS S3 provides a secure bucket where each newly generated composite JSON object can be written to by the system and downloaded by the user.

TESTING

ScalaTest

ScalaTest is a Scala-based testing framework which offers a variety of testing styles (Flat Spec, Fun Spec, Feature Spec, etc.). These allow for full unit test coverage.

Protractor

Protractor is an end-to-end (functional) test framework, designed specifically for AngularJS applications. Built on top of WebDriverJS, Protractor is designed to run tests using a real browser.

Protractor afforded me the ability to mimic user interaction and execute a variety of automated tests.

DEPLOYMENT

Docker

Docker is a packaging/deployment technology which allows for an application to be packaged with all of its code, dependencies, runtime, etc into a single container which can then be easily mounted to a server.

Using Docker in my deployment strategy greatly improves my architecture in terms of speed and scalability and reduces financial overheads.

SYSTEM

Requirements Specification

The following requirements specification outlines a variety of requirements needed in order to deliver the desired system. These requirements are mainly categorised into functional and non-functional requirements. However, considerations have been made in terms of data, environmental user, usability, security, performance and availability requirements.

REQUIREMENTS CATEGORISATION

As this system is composed of a suite of microservices, I felt that it was important to separate the functional and non-functional requirements out into two categories:

1. System level requirements (functional and non-functional)
2. Service level requirements (functional and non-functional)

The reason for this is that each service should be seen as its own standalone application which has its own set of requirements. When those services are then combined to form a system, that system will have its own requirements outlining how it should use the available services.

It is important to note that the system level requirements were derived from the requirements elicitation process conducted during the research stage of this project (see Appendix C), whereas all service level requirements were developed independently.

Please note that, in terms of service level requirements, the following abbreviations are used in place of the full service names:

GTS	Geo Tagging Service	**PCS**	Process Controller Service
SS	AWS Storage Service	**BBDS**	Base Business Details Service

Functional Requirements

SYSTEM LEVEL REQUIREMENTS

Below is a list of all system level functional requirements:

1. The system initialises Spray IO REST/HTTP Integration Layer.
2. The system must provide the user with an interface to interact with each of the systems' composite microservices as well as a combination of those microservices.
3. The system must allow the user to retrieve data based on geographical location.
4. The system must record data measuring the process/response time of each service.
5. The system must provide an analytics dashboard to visualise the system's performance metrics.
6. The system must write the generated data to a downloadable file.
7. The system must display the generated data to the user via the user interface upon successful completion of a service request.

Req-1: REST/HTTP Integration Layer

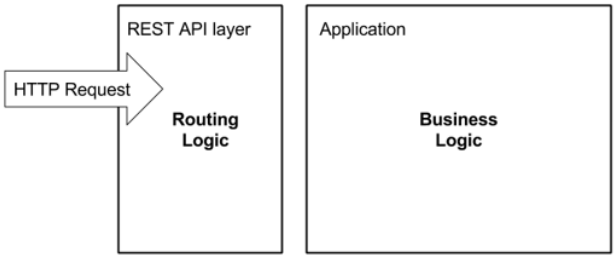
Description and Priority
Each composite service in the system must provide an integration layer which can process incoming HTTP requests and broker the requested service/payload between the service producer and service consumer.

This integration layer is required for any Scala-based service.

This layer will be appended to the service as a form of interface.

This layer will also be responsible for starting a lightweight HTTP server (spray-can) and initialising Akka Actors to handle concurrent threads.

Priority: High

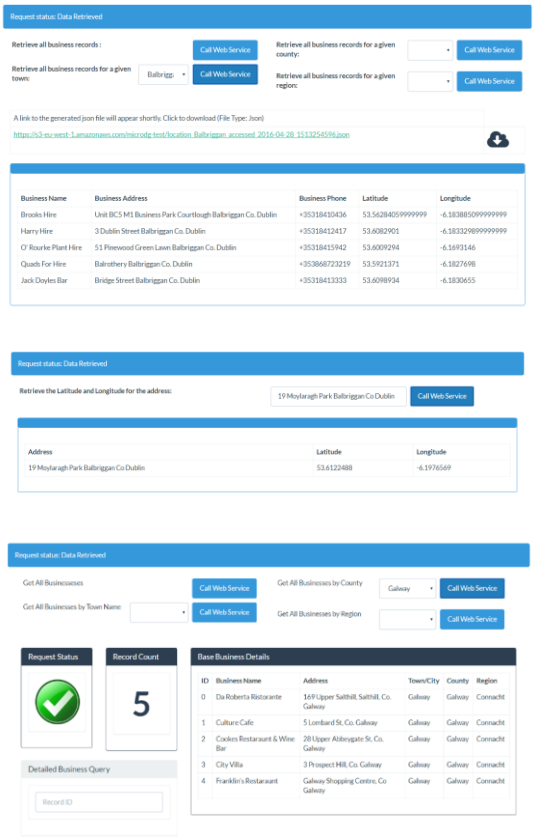


Req-2: Service User Interface

Description and Priority
Each service must have its own user interface/dashboard which provides the user with the ability to retrieve data based on input parameters. Each service should limit the user's input by way of dropdown menus.

Once a service has been successfully executed and a JSON representation of a data object has been returned, that data should be displayed back to the user in the form of a Bootstrap table.

Priority: High



Req-3: Geolocation data retrieval

Description and Priority
The system must present a service which allows the user to retrieve geolocation data pertaining to a given address.
The system must have the ability to receive a human readable address via user input and return only the latitude and longitude coordinates.

Priority: High

Req-4: Runtime metrics

Description and Priority
The system should have the ability to record the runtime duration of each service call and present it back to the user along with additional meta-data such as a date stamp and the number of records returned.

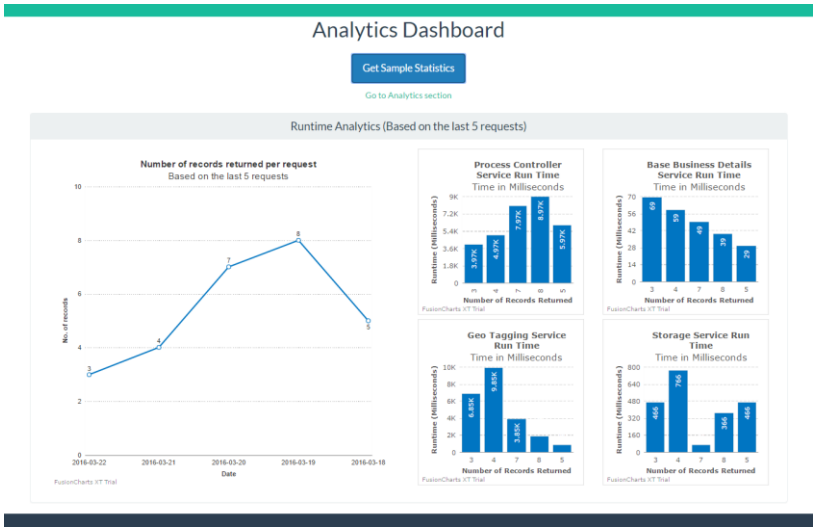
Priority: Low

Req-5: Analytics dashboard

Description and Priority
The system should present the user with a dashboard to chart out statistical data pertaining to service runtimes.

Please see a sample layout below:

Priority: Low



Req-6: Provide downloadable file

Description and Priority
The system should have the ability to write the generated composite data object to a JSON file. That file should be stored on a remote cloud storage service and be available for downloading by the user via a hyperlink.

Priority: Medium

Req-7: Composite data object visualisation

Description and Priority
The system should have the ability to display the content of the generated composite object back to the user via the user interface.

Priority: Medium

SERVICE LEVEL FUNCTIONAL REQUIREMENTS

Below is a list of service level functional requirements:

1. BBDS - Data Retrieval and Serialisation
2. BBDS - User Interface (Dashboard Population)
3. GTS - Retrieve geo coordinates for a given address
4. GTS - User Interface (Dashboard Population)
5. PCS - Generate composite data object
6. PCS - Record runtime statistics
7. SS - Save composite data object to an Amazon S3 bucket

Req-1: BBDS - Data Retrieval and Serialisation

Description and Priority
The service must retrieve data pertaining to business details from an in-memory data collection. The retrieved details should be processed and serialised into an appropriate data interchange format so that the output from this service can be received by the service consumer.

Priority: High

Actors
End users (EU) will have access to this service via the service gateway.

Preconditions
Persistent data exists and is accessible to the service.
The host server is running.

Activation
The service is activated via the following REST/HTTP calls:
GET http://hostdomain/baseBusinesses/all/details
GET http://hostdomain/baseBusinesses/all/details/towns/{param}
GET http://hostdomain/baseBusinesses/all/details/counties/{param}
GET http://hostdomain/baseBusinesses/all/details/regions/{param}
GET http://hostdomain/baseBusinesses/{param}/detail

Main Flow
1. The EU or system makes a HTTP GET request directly to the service requesting all records or subset of records from the in-memory collection (see E1).
2. The Akka actor system is initialised and handles the request along with all subsequent requests.
3. The correct request route is identified and the code body is executed.
4. The route body calls the retrieval function
    a. A query is made to the in-memory data collection.
    b. The query returns a collection (list) of records (can be empty).
    c. The returned list of records is then serialised into a JSON format.
    d. The JSON representation of the collection of queried records is returned to request route.
5. The request route returns the JSON formatted collection to the service consumer via HTTP.

Exceptional Flow
E1: Server Not Running
1. The EU receives a 401 Error Message

Termination: The JSON payload (can be empty) has been returned to the service consumer.

Post Condition: The service enters a wait state.

Req-2: BBDS - User Interface (Dashboard Population)

Description and Priority
The service must be accessible via a standalone user interface (UI). The UI must consist of a button which, when clicked, makes a HTTP GET request to the Base Business Details Service. The UI must parse the returned JSON object and dynamically populate a dashboard.

Desirable features include:
- A table of records
- A record count
- A request status indicator
- A feature to inspect individual records based on a parameter value

Priority: Medium

Actors
End users (EU) will interact directly with this UI.

Preconditions
The server hosting the service is running.

Activation
The user accesses the BBDS-dashboard.html file via the service gateway.

Main Flow
1. The EU clicks on the button named "Call Web Service" (see E1).
2. A HTTP GET request is made to http://hostdomain/baseBusinessDetails/all/details...
3. A JSON string is returned and a number of $scope variables are set:
    a. The list of records is stored as a $scope variable named: bbds_business_list
    b. The number of returned records is stored in a $scope variable named: count
    c. The $scope variable representing the request status is set: status_image
4. The dashboard is automatically populated by the values stored in the $scope variables (see A1).

Alternative Flow
A1: Detailed record view
1. The EU enters an integer value (record ID) into the "Detailed Business Query" input form field.
2. The panel positioned below the input field is populated with further details pertaining to the business with the record ID matching the input parameter (see E1).

Exceptional Flow
E1: Server Not Running
1. The button click will not produce a response.

E2: No record ID matching input value
1. No detailed record is displayed.

Termination: The dashboard is populated.

Req-3: GTS – Retrieve Geo Coordinates for a given address

Description and Priority
The service must retrieve the latitude and longitude coordinates for a given address. The user must have the ability to pass a human readable address as a parameter to the service and retrieve the correct geo coordinates by piggy-backing off the Google Geocoding Web Service. The response must be presented to the user as a JSON object.

Priority: High

Actors
End users (EU) will have access to this service via the service gateway.

Preconditions
The host server is running.
The Google Geocoding Web Service is running.

Activation
The service is activated via the following REST/HTTP calls:
GET http://hostdomain/geoTaggings/addresses/{param}
GET http://hostdomain/geoTaggings/addresses/latitudes/{param}
GET http://hostdomain/geoTaggings/addresses/longitudes/{param}

Main Flow
1. The EU or system makes a HTTP GET request directly to the service by passing a human readable address string as a parameter (see E1).
2. The Akka actor system is initialised and handles the request along with all subsequent requests.
3. The correct request route is identified and the code body is executed.
4. The route body calls the geo tagging function
    a. A HTTP GET request is made to the Google Geocoding Web Service.
    b. The JSON response is parsed and the latitude and longitude coordinates are extracted.
    c. Both of the above values are used to create a new Geo Coordinates object.
    d. This new object is then serialised into a JSON format.
    e. The JSON representation of the object is returned to request route.
5. The request route returns the JSON formatted object to the service consumer via HTTP.

Exceptional Flow
E1: Server Not Running
1. The EU receives a 401 Error Message

Termination: The JSON payload has been returned to the service consumer.

Post Condition: The service enters a wait state.

Req-4: GTS - User Interface (Dashboard Population)

Description and Priority
The service must be accessible via a standalone user interface (UI). The UI must consist of a button which, when clicked, makes a HTTP GET request to the Geo Tagging Service. The UI must parse the returned JSON object and populate a single row table.

Priority: Medium

Actors
End users (EU) will interact directly with this UI.

Preconditions
The server hosting the service is running.
The Google Geocoding Web Service is running.

Activation
The user accesses the GTS-dashboard.html file via the service gateway.

Main Flow
1. The EU clicks on the button named "Call Web Service" (see E1 and E2).
2. A HTTP GET request is made to http://hostdomain/geoTaggings/addresses/{param}
3. A JSON string is returned and a number of $scope variables are set:
    a. The geo coordinates object is stored as a $scope variable named: g_cords_list.
    b. The query address is also made available as a $scope variable called: query_address.
4. The dashboard is automatically populated by the values stored in the $scope variables (see A1).

Exceptional Flow
E1: Server Not Running
1. The button click will not produce a response.

E2: Google Geocoding Web Service is down
1. The button click will not produce a response.

Termination: The dashboard is populated.

Req-5: PCS – Generate a composite data object

Description and Priority
The service must retrieve specific data from both the Base Business Details Service and then Geo Tagging Service and merge them in order to create a new composite data object. This object will comprise of the name, address, phone number and geo coordinates for each business returned.

Priority: High

Actors
End users (EU) will have access to this service via the service gateway.

Preconditions
The host server and Google Geocoding Web Service are running.

Activation
The service is activated via the following REST/HTTP call:
GET http://hostdomain/api/processControllers/processA/locationType/{param}/locationValue{param}

Main Flow
1. The EU or system makes a HTTP GET request directly to the service requesting all records or subset of records based on the query parameters (see E1 and E2).
2. The Akka actor system is initialised and handles the request along with all subsequent requests.
3. The correct request route is identified and the code body is executed.
4. The route body calls the retrieval function
    a. A HTTP GET request is made to the Base Business Details Service.
    b. The required values are parsed out and stored in individual collections.
    c. A HTTP GET request is made to the Google Geocoding Web Service.
    d. The latitude and longitude coordinate values are parsed out and stored in individual collections.
    e. All five collections are merged together to form a composite data object.
    f. This new object is then serialised into a JSON format.
    g. The JSON representation of the object is returned to the request route.
5. The request route returns the JSON formatted object to the service consumer via HTTP.

Exceptional Flow
E1: Server Not Running
1. The button click will not produce a response.

E2: Google Geocoding Web Service is down
1. The button click will not produce a response.

Termination: The JSON payload has been returned to the service consumer.

Post Condition: The service enters a wait state.

Req-6: PCS – Record runtime statistics

Description and Priority
Throughout the execution of the process controller, the runtime duration of that service as well as the services it calls should be recorded and pushed as an object to a queue. There runtime statistic objects can then be queried by a separate process.

Priority: Medium

Actors
System: The process controller (PCS)

Preconditions
The host server is running.

Activation
The service is activated via the following REST/HTTP call:
GET http://hostdomain/api/processControllers/processA/locationType/{param}/locationValue{param}/withStats

Main Flow
1. The EU or system makes a HTTP GET request directly to the service requesting all records or subset of records based on the query parameters (see E1 and E2).
2. The Akka actor system is initialised and handles the request along with all subsequent requests.
3. The correct request route is identified and the code body is executed.
4. The route body calls the retrieval function
    a. The system records the start time for the PCS.
    b. The system records the start time for the BBDS.
    c. The system records the end time for the BBDS.
    d. The system records the start time for the GTS.
    e. The system records the end time for the GTS.
    f. The system records the start time for the SS.
    g. The system records the end time for the SS.
    h. The system records the end time for the PCS.
    i. The system retrieves the number of records in the composite object.
    j. The system retrieves the current system date stamp.
    k. The system assigns the difference between the start and end times of each service to new variables.
    l. The difference variables, date stamp and record count are all stored as a new object.
    m. This object is then pushed to the statistics queue.

Exceptional Flow
E1: Server Not Running
1. The button click will not produce a response.

E2: Google Geocoding Web Service is down
1. The button click will not produce a response.

Termination
The runtime statistics object is pushed to the queue.

Post Condition
The queue has been updated with a new object.

Req-7: SS – Save composite data object to an Amazon S3 bucket

Description and Priority
Prior to the end of the execution of the process controller, the composite data object should be written to a remote file in an Amazon S3 bucket. This process should also generate a hyperlink which the user can then use to download the file to their local machine.

Priority: High

Actors
System: The process controller (PCS)

Preconditions
The host server is running.

Activation
The service is activated via the following REST/HTTP call:
GET http://hostdomain/api/processControllers/processA/locationType/{param}/locationValue{param}/withStats

Main Flow
1. The EU or system makes a HTTP GET request directly to the service requesting all records or subset of records based on the query parameters (see E1 and E2).
2. The Akka actor system is initialised and handles the request along with all subsequent requests.
3. The correct request route is identified and the code body is executed.
4. The route body calls the retrieval function
    a. The system generates a URL which will serve as the destination and file name of the new composite data object.
    b. The system uses the AWS transfer manager to write the composite data object to the S3 bucket.

Exceptional Flow
E1: Server Not Running
1. The button click will not produce a response.

E2: Google Geocoding Web Service is down
1. The button click will not produce a response.

Termination: A copy of the composite object has been written to persistent cloud storage.

Post Condition: The composite object is available to be downloaded by the user via the URL generated by the process controller.

Non-Functional Requirements

SYSTEM LEVEL NON-FUNCTIONAL REQUIREMENTS

Below is a list of all system level non-functional requirements:

1. The system should offer the user the option to search using a range of options (such as town, city and region).
2. The system should utilise a microservices architecture.
3. The system should be scalable.
4. Communication between microservices and the user interface should be handled via REST/HTTP messages.
5. The system should provide the user with a link to view the generated data file.
6. The system's composite microservices should respond to requests using JSON.

Performance and Availability Requirements

The system should be scalable so that it can meet the demand of high volumes of traffic during peak times.

The system should be developed using horizontally scaling technologies such as Akka in order to increase mass concurrent access to the system and reduce possible downtime.

Not all services will require the same level of performance and availability, for example the service used to retrieve the geo coordinates information will require a higher rate of performance and availability than that of the service retrieving the base business details.

Security Requirements

The service responsible for interfacing with the Amazon S3 bucket will use basic client/secret key authentication between service accounts (server-to-server).

User Requirements

This system has one intended "class of user". I have outlined the specific user requirements below:

END USER (EU)

1. As an EU, I want the ability to compile data from multiple sources into the one data object.
2. As an EU, I want the ability to view and download the generated data in the form of a JSON file.
3. As an EU, I want the system to be simple and intuitive to use.
4. As an EU, I want the system to be fast.

Design and Architecture

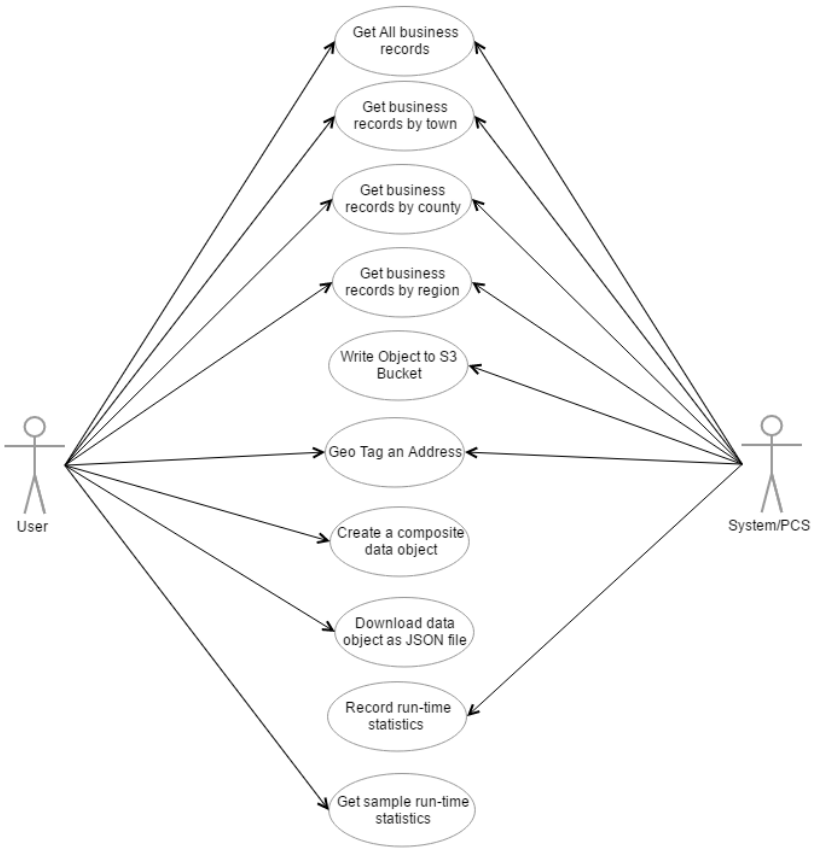
This system was designed and developed using the microservices architecture pattern. The purpose of this pattern is to functionally decompose the traditional monolithic system design into standalone services which can act autonomously and can be scaled and deployed independently.

Each microservice runs its own process in its own environment and does not share storage or data sources. Each microservice also uses lightweight communication mechanisms (typically a REST/HTTP resource API).

Each service is designed to do one thing but can communicate with other services via a HTTP integration layer to form a larger system (suite of services).

SYSTEM ARCHITECTURE DIAGRAM

SYSTEM USE CASE MODEL



SERVICE OVERVIEW

Base Business Details Service	A service which retrieves a list of businesses and their base details from an in-memory data collection.
Geo Tagging Service	A service which retrieves the latitude and longitude coordinates for each address passed to it.
AWS Storage Service	A service for the persistent storage of composite data objects (JSON).
Process Controller Service	A process controller which acts as a proxy between the UI and the individual web services.

ADVANTAGES OF THE MICROSERVICES ARCHITECTURE

The Microservices architecture pattern boasts many advantages over the traditional Monolithic pattern including:

- Loose coupling
- Firm module boundaries and separation of concerns
- Independently deployable and scalable
- Faster deployment cycles
- Language independence

Due to these advantages, many large companies have migrated away from a monolithic architecture to a microservices one over the past several years including Netflix, Amazon and eBay.

CHARACTERISTICS OF THE MICROSERVICES ARCHITECTURE

In their 2014 article on Microservices, Lewis and Fowler[4] proposed that, although there exists no strict definition of what a microservice is, the following nine constraints are characteristic of a microservice:

1. Componentisation via Services
2. Organised around Business Capabilities
3. Products not Projects
4. Smart Endpoints and Dumb Pipes
5. Decentralised Governance
6. Decentralised Data Management
7. Infrastructure Automation
8. Design for Failure
9. Evolutionary Design

[4] Lewis, James, and Fowler, Martin. "Microservices". *martinfowler.com*. N.p., 2014. Web. 31 Jan. 2016.

Componentisation via Services	A software component is considered to be a unit of software that can replaced, upgraded and deployed independently. A component must comprise of the necessary functions and resources to carry out a single piece of work (example: checking a customer's available credit in an online store) and that functionality should be exposed as a service.
Organised around Business Capabilities	Typically, development teams are separated in accordance to the tier or technology layer that they operate on (example: UI Developers, Server Side Engineers and Database Engineers). The microservices pattern proposes that teams are organised into cross functional organisations whereby each team will have the personnel to handle the full technology stack.
Products not Projects	Each development team is responsible for developing a product as well as its long-term maintenance. For example, a team that develops a "Shopping Basket" feature for an online store assumes long running ownership of that feature and is responsible for its health.
Smart Endpoints and Dumb Pipes	Each service is responsible for the full processing of data at the endpoint and does not rely on sophisticated Enterprise Service Buses (ESB).
Decentralised Governance	Microservices are language and technology independent from one another. The advantage of this is the ability to use the right tool for the job. Constituent services in a system can be written in C++, Java, Scala, Ruby, etc. Once a service can communicate over HTTP using a standard data-interchange format such as JSON, the development language is open.
Decentralised Data Management	Databases should not be shared between services. This allows for independent views of domain objects and a choice of persistent storage options.
Infrastructure Automation	The concept of microservices supports continuous deployment. Each service should have its own pipeline handling automated testing and deployment.
Design for Failure	A microservices system must be designed in a way that it can tolerate failure of its constituent services.
Evolutionary Design	Replace ability and upgradability are key features of a software component/module. When dividing a system into a suite of services, ones that experience high rates of change should be separated from the ones that experience low rates of change.

IMPLEMENTATION

Introduction to the MicroDG microservices

As mentioned in the previous section, the MicroDG system is comprised of four constituent microservices:

- Base Business Details Service (BBDS)
- Geo Tagging Service (GTS)
- AWS Storage Service (SS)
- Process Controller Service (PCS)

Each of these services is primarily written in Scala and exposes a HTTP integration layer using Spray. Due to this fact, all of the services share some common features. I have decided to highlight these features in a separate section so as to not unnecessarily repeat any code snippets.

Common Feature: Spray HTTP Integration Layer

Each constituent service exposes a lightweight REST/HTTP integration layer (Spray + Akka) in order to connect each Scala application to the user (or to other systems). The specific HTTP server used here is called "spray-can", it is fully asynchronous and has the ability to handle thousands of concurrent connections. This alone greatly increases the scalability of each service.

Each service API endpoints, and the functions they call, are different. However, they all share common code blocks. These are listed below:

1. Initialise the Akka Actor System to handle concurrent requests

```
implicit val actorSystem = ActorSystem()
```

src/main/scala/com/server/

2. Defining custom directives for:
    a. Setting the media type response header to JSON
    b. Including CORS (Cross Origin Resource Sharing)

```
def getJson(route: Route) = get{
    respondWithMediaType(MediaTypes.`application/json`){
    route
   }
}
```

src/main/scala/com/server/

3. Defining routes and their actions

```
lazy val helloRoute = get {
   cors{
      path("hello") {
         complete {
             "Welcome to the Base Business Details Service "
              }
           }
       }
   }
```

src/main/scala/com/server/

4. Starting the lightweight HTTP server (spray-can)

```
startServer(interface = "localhost", port = 8080) {
   helloRoute~
   listAllRoute
}
```

src/main/scala/com/server/

Common Feature: Cross Origin Resource Sharing

CORS (Cross Origin Resource Sharing) is a mechanism for allowing resources to be requested from outside of the origin domain. It is essential to have CORS in place when wishing to expose data between two objects via an API.

Fortunately, there exists a ready-made Scala directive that provides CORS functionality, which I was able to use for each service.

Please note that the following code snippet is not my original work and I have referenced the original author accordingly.[5]

```
package com.support

import spray.http.{HttpMethods, HttpMethod, HttpResponse, AllOrigins}
import spray.http.HttpHeaders._
import spray.http.HttpMethods._
import spray.routing._

// see also https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS
trait CORSSupport {
  this: HttpService =>

  private val allowOriginHeader = `Access-Control-Allow-Origin`(AllOrigins)
  private val optionsCorsHeaders = List(
    `Access-Control-Allow-Headers`("Origin, X-Requested-With, Content-Type, Accept, Accept-
Encoding, Accept-Language, Host, Referer, User-Agent"),
    `Access-Control-Max-Age`(1728000))

  def cors[T]: Directive0 = mapRequestContext { ctx => ctx.withRouteResponseHandling({
    //It is an option requeset for a resource that responds to some other method
    case Rejected(x) if (ctx.request.method.equals(HttpMethods.OPTIONS) &&
!x.filter(_.isInstanceOf[MethodRejection]).isEmpty) => {
      val allowedMethods: List[HttpMethod] =
x.filter(_.isInstanceOf[MethodRejection]).map(rejection=> {
        rejection.asInstanceOf[MethodRejection].supported
      })
      ctx.complete(HttpResponse().withHeaders(
        `Access-Control-Allow-Methods`(OPTIONS, allowedMethods :_*) ::  allowOriginHeader
::
         optionsCorsHeaders
      ))
    }
  }).withHttpResponseHeadersMapped { headers =>
    allowOriginHeader :: headers

  }
  }
}
```

src/main/scala/com/support/CORSSupport.scala

[5] Raya, Jose. "CORS Directive For Spray". *Gist*. N.p., 2016. Web. 27 Apr. 2016.

Microservice: Base Business Details Service

SERVICE OVERVIEW

The Base Business Details Service (BBDS) is a microservice for the querying and retrieval of data pertaining to commercial businesses, from an in-memory data store. This service exposes a RESTful API which allows the user to retrieve data based on query parameters such as town name, county and region.

Source Code Repository

https://github.com/microdg/BETA-Base-Business-Details-Service.git

COMPOSITE TECHNOLOGIES

- Scala
- Spray (spray-can)
- Akka

CODE BASE OVERVIEW

Directory	**File**	**Description**
/	build.sbt	Standard SBT file listing all required dependencies and resolvers
src/main/scala/com/server	BaseBusinessDetailsService.scala	HTTP integration layer exposes RESTful API endpoints. The server object is activated on boot
src/main/scala/com/services	BBDRetrieval.scala	Contains all querying functions to interact with the in-memory data store
src/main/scala/com/support	CORSSupport.scala	A custom directive to enable Cross Origin Resource Sharing
	LoggingSupport.scala	Contains a function for logging service calls
src/main/scala/com/models	Business.scala	Defines the data model for a business object and provides a collection of objects for querying

API ENDPOINTS

Please note that all URIs are based on the scenario that the service is being run locally.

Endpoint Reference	baseBusinesses/all/details
Description	Retrieve a JSON representation of all business records in memory
URI:	http://localhost:8081/baseBbusinesses/all/details
HTTP verb	GET
Parameters	None
Pre Conditions	None
Post Conditions	The data is returned in JSON format for rendering and visualisation

Endpoint Reference	baseBusinesses/all/details/towns/<town_name>
Description	Retrieve a JSON representation of all business records in memory based on a given town
URI:	http://localhost:8081/baseBusinesses/all/details/towns/<town_name> Example: http://localhost:8081/baseBusinesses/all/details/towns/Skerries
HTTP verb	GET
Parameters	Town Name (Example: Skerries) Data Type: Segment (String)
Pre Conditions	None
Post Conditions	The data is returned in JSON format for rendering and visualisation

Endpoint Reference	baseBusinesses/all/details/counties/<county_name>
Description	Retrieve a JSON representation of all business records in memory based on a given county
URI:	http://localhost:8081/baseBusinesses/all/details/counties/<county_name> Example: http://localhost:8081/baseBusinesses/all/details/counties/Dublin
HTTP verb	GET
Parameters	County Name (Example: Dublin) Data Type: Segment (String)
Pre Conditions	None
Post Conditions	The data is returned in JSON format for rendering and visualisation

Endpoint Reference	baseBusinesses/all/details/region/<region_name>
Description	Retrieve a JSON representation of all business records in memory based on a given region
URI:	http://localhost:8081/baseBusinesses/all/details/regions/<region_name> Example: http://localhost:8081/baseBusinesses/all/details/regions/Leinster
HTTP verb	GET
Parameters	Region Name (Example: Leinster) Data Type: Segment (String)
Pre Conditions	None
Post Conditions	The data is returned in JSON format for rendering and visualisation

Endpoint Reference	baseBusinesses/all/count
Description	Retrieve a total count of the number of records stored in memory
URI:	http://localhost:8081/baseBusinesses/all/count
HTTP verb	GET
Parameters	None
Pre Conditions	None
Post Conditions	The data is returned in JSON format for rendering and visualisation

Endpoint Reference	baseBusinesses/<record_ID>/details
Description	Retrieve a JSON representation of an individual business record based on a given ID
URI:	http://localhost:8081/baseBusinesses/<record_ID>/details Example: http://localhost:8081/baseBusinesses/1/details
HTTP verb	GET
Parameters	Record ID: (Example: 1) Data Type: Integer
Pre Conditions	None
Post Conditions	The data is returned in JSON format for rendering and visualisation

SAMPLE RESPONSE

The following JSON object is a sample response from the microservice based on the following:

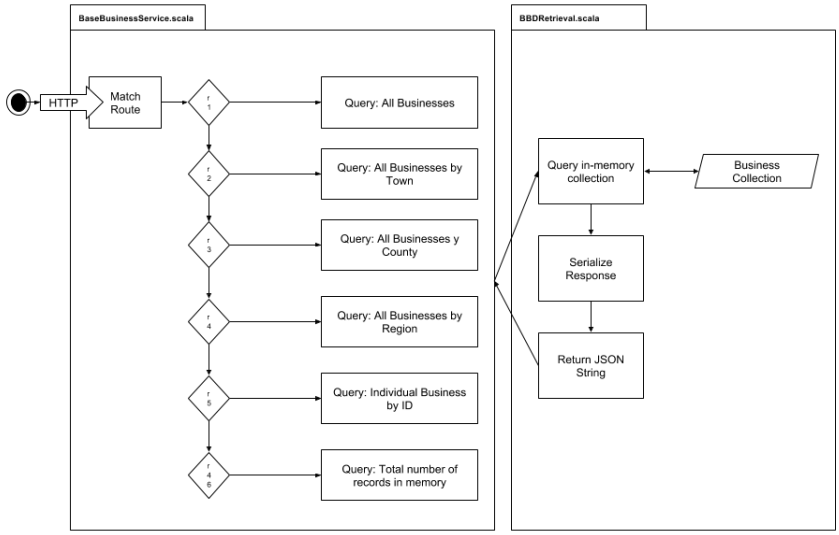
Endpoint: baseBusinesses/<record_ID>/details
Parameter: 1


```
[
  {
    "id": 1,
    "name": "Landmark Estates",
    "channel": "property",
    "channel_type": "estate agents",
    "phone": "+35318414726",
    "address": "21 Drogheda Street Balbriggan Co. Dublin",
    "town": "Balbriggan",
    "county": "Dublin",
    "region": "Leinster"
  }
]
```

PROCESS FLOW DESCRIPTION

1. The BBDS receives a HTTP GET request from the End User (EU) via the User Interface (UI).
2. The BBDS parses the available routes and executes the request when the correct route has been identified/matched.
3. The BBDS queries the in-memory data collection.
4. The BBDS serialises the response object to a JSON format.
5. The BBDS returns the JSON string to the UI.
6. The BBDS enters a wait state.

PROCESS FLOW DIAGRAM



FUNCTION/METHOD HIGHLIGHT

Query Functions

The Base Business Details Service contains a number of query functions used to retrieve specific data from the in-memory collection pertaining to a parameter passed by the user.
In the following example, I will demonstrate a function which retrieves a list of all business records that have a town name matching the one passed by the user.

```
//Query: Retrieve all records based on a town name
def businessesByTown(town: String) : String = {
        val businesses = Business.businesses
        val businessesByTown = for(b <- businesses if b.town == town) yield b
        val jsonString = write(businessesByTown)
        LoggingSupport.serviceRequestlog1("Business By Town", "Town")
        return jsonString
}
```

src/main/scala/com/services/BBDRetrieval.scala

1. All business records are stored in a local object called “businesses”.
2. Iterate over each business in the businesses object. If the business has a town name matching the one passed to the function by the user, add that business to a new object called businessesByTown using the Scala yield function.
3. Convert the businessesByTown to JSON.
4. Log the service request using the custom support logger.
5. Return the JSON object to the HTTP integration layer.

Count Function

Here is a simple function to return the number of records stored in the collection as an integer:

```
//Query: Retrieve a count of all records
def businessListCount() : Int = {
        val businesses = Business.businesses
        val count = businesses.length
        return count
}
```

src/main/scala/com/services/BBDRetrieval.scala

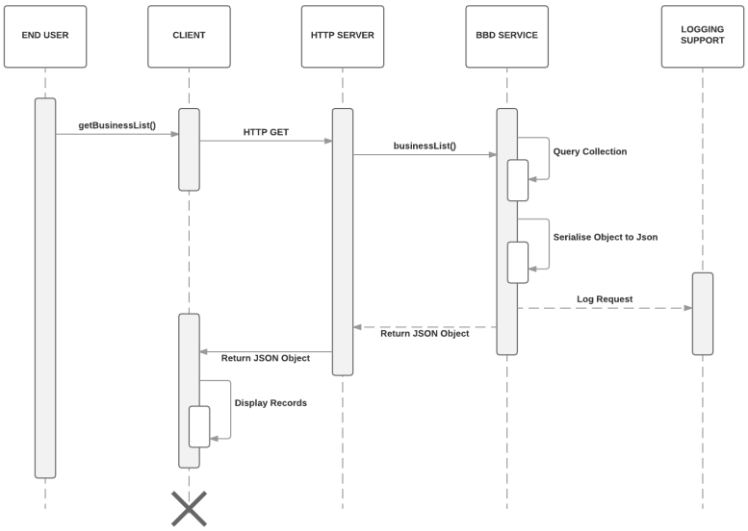
1. All business records are stored in a local object called “businesses”.
2. Use the Java length function to retrieve the size of the collection.
3. Return the count object to the HTTP integration layer.

CLIENT/SERVER SEQUENCE DIAGRAM

The following sequence diagram shows the interaction between the client and server for the following service request:

- http://localhost:8081/baseBusinesses/all/details

This sequence is shared amongst all BBDS requests.



CODE COVERAGE

A suite of unit tests can be found under the src/test/scala/com/services directory. These tests employ the ScalaTest framework and are written using the "FlatSpec" trait. These unit tests cover the query functions and ensure that they deliver the expected output. A detailed breakdown of these tests can be found under the Unit Testing (White Box) section (page 76).

Microservice: Geo Tagging Service

SERVICE OVERVIEW

The Geo Tagging Service (GTS) is a microservice for tagging a human readable address with its corresponding latitude and longitude coordinates. This service exposes a RESTful API and responds to the user's request with a JSON object.

Source Code Repository

https://github.com/microdg/BETA-Geo-Tagging-Service.git

COMPOSITE TECHNOLOGIES

- Scala
- Spray (spray-can)
- Akka
- Google Geocoding API

CODE BASE OVERVIEW

Directory	**File**	**Description**
/	build.sbt	Standard SBT file listing all required dependencies and resolvers
src/main/scala/com/server	GTServer.scala	HTTP integration layer exposes RESTful API endpoints. The server object is activated on boot
src/main/scala/com/services	GTRetrievalService.scala	Contains the main process controller functions for the service and handles the connection to the external third party web service
src/main/scala/com/support	CORSSupport.scala	A custom directive to enable Cross Origin Resource Sharing
	LoggingSupport.scala	Contains a function for logging service calls
	GTRetrievalHelper.scala	A series of helper functions to handle URL construction as well as data parsing and formatting

API ENDPOINTS

Please note that all URIs are based on the scenario that the service is being run locally.

Endpoint Reference	geoTaggings/addresses/<address_parameter>
Description	Retrieve a JSON representation of the latitude and longitude coordinates for a given address
URI:	http://localhost:8082/geoTaggings/addresses/<address_parameter>
HTTP verb	GET
Parameters	Address (Example: 10 Dublin Street, Balbriggan, Co. Dublin) Data Type: Segment (String)
Pre Conditions	None
Post Conditions	The data is returned in JSON format for rendering and visualisation

Endpoint Reference	geoTaggings/addresses/latitudes/<address_parameter>
Description	Retrieve a JSON representation of the latitude coordinate for an address
URI:	http://localhost:8082/ geoTaggings/addresses/latitudes/<address_parameter>
HTTP verb	GET
Parameters	Address (Example: 10 Dublin Street, Balbriggan, Co. Dublin) Data Type: Segment (String)
Pre Conditions	None
Post Conditions	The data is returned in JSON format for rendering and visualisation

Endpoint Reference	geoTaggings/addresses/longitudes/<address_parameter>
Description	Retrieve a JSON representation of the longitude coordinate for an address
URI:	http://localhost:8082/ geoTaggings/addresses/longitudes/<address_parameter>
HTTP verb	GET
Parameters	Address (Example: 10 Dublin Street, Balbriggan, Co. Dublin) Data Type: Segment (String)
Pre Conditions	None
Post Conditions	The data is returned in JSON format for rendering and visualisation

SAMPLE RESPONSE

The following JSON object is a sample response from the microservice based on the following:

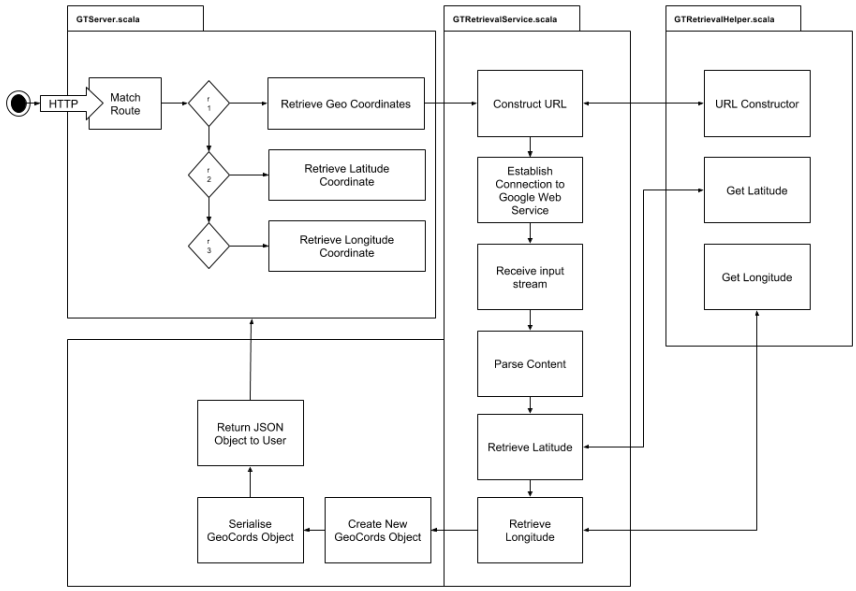
Endpoint: geoTaggings/addresses/<address_parameter>
Parameter: 10 Dublin Street, Balbriggan, Co. Dublin


```
[
  {
    "lat": "53.6078703",
    "lng": "-6.1830988"
  }
]
```

PROCESS FLOW DESCRIPTION

1. The GTS receives a HTTP GET request from the End User (EU) via the User Interface (UI).
2. The GTS parses the available routes and executes the request when the correct route has been identified/matched.
3. The GTS accepts the address parameter passed by the user and passes it to the "URL Constructor" helper function.
    1. The function reformats the address string, replacing blank spaces with a "+" symbol. This is the format expected by the Google Geocoding Web Service.
    2. The function constructs a URL comprising of the newly formatted address parameter, the Google Geocoding API endpoint and a valid API key.
4. The GTS uses the new URL to connect to the Google Geocoding Web Service.
5. The JSON response from the Geo Coding Web Service is read via an input stream and stored in a new string object (content).
6. The content object is parsed using the third party JSON parser (lift web).
7. The parsed content object is then passed into two separate functions in order to extract the latitude and longitude values separately.
8. A new "GeoCords" object is instantiated with the latitude and longitude values passed into the case class constructor.
9. This new "GeoCords" object is then serialised to JSON.
10. The GTS returns the JSON string to the UI.
11. The GTS enters a wait state.

PROCESS FLOW DIAGRAM



FUNCTION/METHOD HIGHLIGHT

URL Constructor

The purpose of this helper function is to take the address parameter passed in by the user and then format it in a way (replace spaces with "+" symbols) that the Google Geocoding API can accept. The function also appends the API key and endpoint to the formatted address so that a complete URL is returned to the user.

```
//Function: A helper function for constructing a a url
 def urlConstructor(address: String) : String = {
     val apiEndpoint = "https://maps.googleapis.com/maps/api/geocode/json?address="
     val apiKey = "&key=AIzaSyCW6S4V0IgkgoXwsgZUtdTTNRzM2zPqe9E"
     val formattedAddress = address.replace(" ", "+")
     val url = apiEndpoint + formattedAddress + apiKey
     return url
 }
```

src/main/scala/com/support/GTRetrievalHelper.scala

1. Set a static endpoint string.
2. Set a static API key string.
3. Use the native replace function to replace all spaces with "+" symbols, with the result being stored as a new string.
4. Concatenate all three strings together to form a new URL string.
5. Return the new URL.

Latitude & Longitude Retriever

There are two functions for parsing out exact values from the JSON object returned by the call to the Google Geocoding API. They only differ from each other by the key they are searching for ("lat" or "lng").

```
//Function: A helper function for parsing out the Latitude value
def getLatitude(json: JValue): String = {

     val jLat = json find {
              case JField("lat", _) => true
              case _ => false
           }

     val sLat = write(jLat)
     val lat =  sLat.substring(6, sLat.length)
     return lat
 }
```

src/main/scala/com/support/GTRetrievalHelper.scala

1. Use the lift web query functionality to find the desired JSON value and store it as an object.
2. Re-serialise the object to JSON.
3. Format the string by removing the characters preceding character six and preceding the character at the length of the string.
4. Return the formatted object as a string.

Geo Coordinates Retrieval

The following function is the main function of this service and is responsible for retrieving the latitude and longitude coordinates for a given address:

```
def getGeoTags_single(address: String, connectTimeout: Int = 4000, readTimeout: Int = 4000,
requestMethod: String = "GET") : String = {

            //Logging: Log the service request
            LoggingSupport.serviceRequestlog1("Geo Tagging Service", address)

            //Construct url using helper function
            val url = GTRetrievalHelper.urlConstructor(address)

            //Establish connection to Googles Geo Coding web service
            val connection = (new URL(url)).openConnection.asInstanceOf[HttpURLConnection]
            println(connection)
            connection.setConnectTimeout(connectTimeout)
            connection.setReadTimeout(readTimeout)
            connection.setRequestMethod(requestMethod)

            val inputStream = connection.getInputStream
            //Parse the connction.InputStream to JSON
            val content = io.Source.fromInputStream(inputStream).mkString

            if (inputStream != null) inputStream.close

            val json = parse(content)

            //Retrieve the desired values
            val lat = GTRetrievalHelper.getLatitude(json)
            val lng = GTRetrievalHelper.getLongitude(json)
            //Create a new object with the retrieved values as paramters
            val geoCords = List[GeoCords](GeoCords(s"$lat",s"$lng"))
            //Convert the objet back to JSON
            val jsonString = write(geoCords)
            //Return JSON Object
            return jsonString
        }
```

src/main/scala/com/services/GTRetrievalService.scala

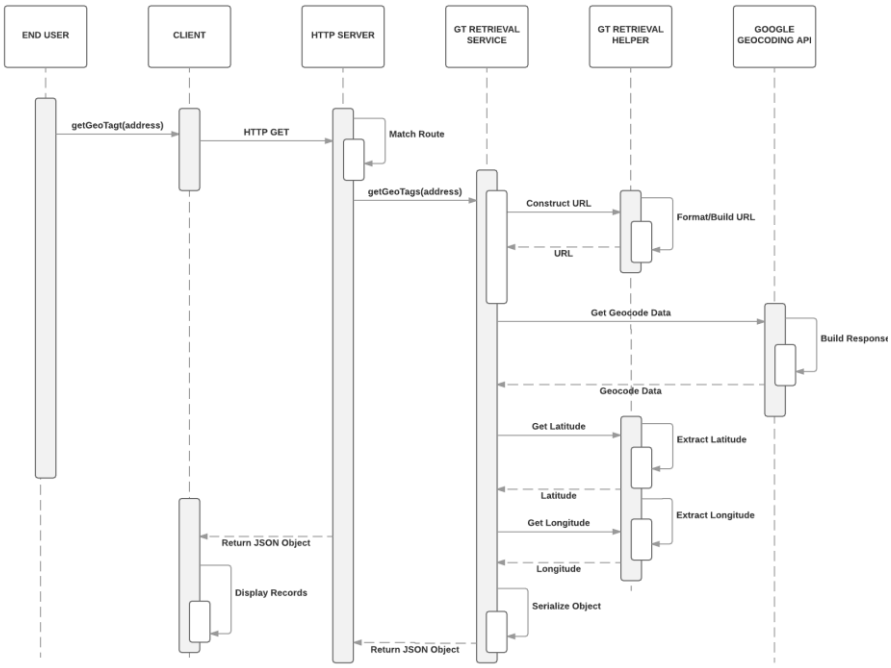
1. Accept the address string as a parameter along with some set parameters used for establishing a connection to the Google Geocoding Web Service.
2. Log the service request using the custom support logger.
3. Construct a URL using the helper URL constructor support function.
4. Establish a connection with the Google Geocoding Web Service.
5. Store the input stream as a string object called content.
6. Close the input stream.
7. Convert the content object to JSON.
8. Use the support/helper functions to retrieve the latitude and longitude coordinates.
9. Create a new object (GeoCords) using the new latitude and longitude values.
10. Convert that new object to JSON.
11. Return the JSON object to the HTTP integration layer.

CLIENT/SERVER SEQUENCE DIAGRAM

The following sequence diagram shows the interaction between the client and server for the following service request:

- http://localhost:8082/geoTaggings/addresses/<address_parameter>

This sequence is shared amongst all GTS requests.



CODE COVERAGE

A suite of unit tests can be found under the src/test/scala/com/services and src/test/scala/com/support directories. These tests employ the ScalaTest framework and are written using the "FlatSpec" trait. These unit tests cover the service and helper functions and ensure that they deliver the expected output. A detailed breakdown of these tests can be found in Unit Tests (White Box) section (page 77).

Microservice: AWS Storage Service

SERVICE OVERVIEW

The AWS Storage Service (SS) is a microservice for storing the composite data object, generated by the process controller, to an Amazon S3 Bucket. Once stored, the user will have the ability to download the object as a JSON file.

Source Code Repository

https://github.com/microdg/BETA-AWS-Storage-Service.git

COMPOSITE TECHNOLOGIES

- Scala
- Spray (spray-can)
- Akka
- Java
- Amazon AWS libraries

CODE BASE OVERVIEW

Directory	File	Description
/	build.sbt	Standard SBT file listing all required dependencies and resolvers
src/main/scala/com/server	SSServer.scala	HTTP integration layer exposes RESTful API endpoints and the server object is activated on boot
src/main/scala/com/services	StorageService.scala	Contains the main function to write the composite object to the remote S3 bucket
src/main/scala/com/support	CORSSupport.scala	A custom directive to enable Cross Origin Resource Sharing
	StorageServiceHelper.scala	A series of helper functions to retrieve time and date stamps

API ENDPOINTS

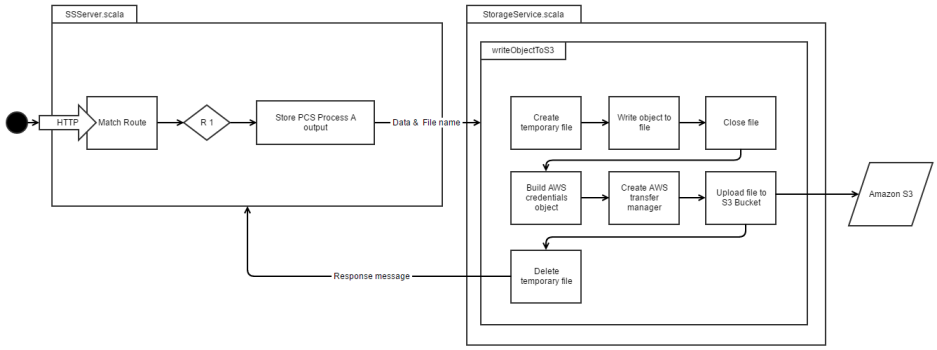
Please note that all URIs are based on the scenario that the service is being run locally.

Endpoint Reference	storageServices/s3/processControllers/processA/withObject/<json_object> /andDestination/<url>
Description	Push the composite JSON object to a designated Amazon S3 bucket
URI:	http://localhost:8084/ storageServices/s3/processControllers/processA/withObject/<json_object> /andDestination/<url>
HTTP verb	GET
Parameters	Data Type: Segment (Composite JSON Object) Data Type: Segment (String)
Pre Conditions	None
Post Conditions	A new object has been added to the remote S3 bucket

PROCESS FLOW DESCRIPTION

1. The SS receives a HTTP GET request from the End User (EU) via the User Interface (UI).
2. The SS parses the available routes and executes the request when the correct route has been identified/matched.
3. The GTS accepts the composite JSON object and the destination URL parameters passed by the user.
4. The SS actions the "writeObjectToS3" function with the above parameters.
5. A new temporary file is created.
6. The native Java Print Writer is used to write the composite data object to the file.
7. An AWS Credentials object is constructed using the access key ID and secret access key.
8. The target S3 bucket is identified by name and the file is then uploaded using the AWS Transfer Manager.
9. The temporary file is then deleted.

PROCESS FLOW DIAGRAM



FUNCTION/METHOD HIGHLIGHT

S3 Object Writer

The following function is the main function of this service and is responsible for writing the composite JSON object to a file and then uploading that file to a remote Amazon S3 bucket:

```
  //Function: Write the composite JSOn object to an S3 Bucket
  def writeObjectToS3(data: String, fileName: String): String = {

        //Here I use Java Printwriter to write object to a temporary file
        val f: File = new File(fileName)
        f.createNewFile()
        val pw: PrintWriter = new PrintWriter(f)
        pw.write(data);
        pw.close();

        //Here I define my AWS Credentials - Do not upload these to GitHub-
        val accessKeyId = "xxxxxxxxxxxxxxxxxxxxx"
        val secretAccessKey = "xxxxxxxxxxxxxxxxxxxxxxxxx"

        //Here I am building the AWS Credentials using the above values
        val cred: AWSCredentials = new BasicAWSCredentials(accessKeyId, secretAccessKey)

        //Here I am uploading the data to the S3 bucket
        val tm: TransferManager = new TransferManager(cred)
        val upload: Upload = tm.upload("microdg-test", fileName, f)
        upload.waitForCompletion();
        println("Upload complete.");

        //Here I delete the temporary file
        f.getAbsoluteFile().delete();

        return s"Find the file at this location: $fileName"
    }
}
```

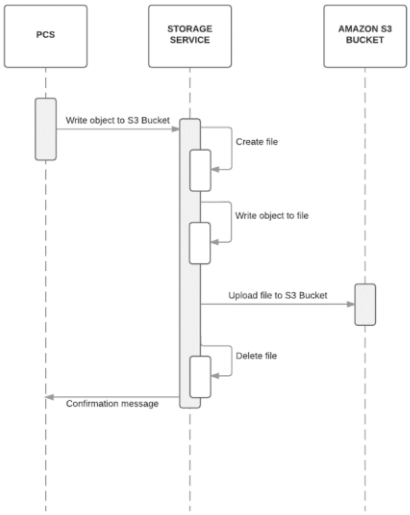
src/main/scala/com/services/StorageService.scala

1. Create a new file.
2. Write the composite JSON object to the file.
3. Create an AWS Credentials object using the access key ID and secret access key.
4. Upload the file to the remote S3 bucket using the AWS Transfer Manager.
5. Delete the file.

CLIENT/SERVER SEQUENCE DIAGRAM

The following sequence diagram shows the interaction between the client and server for the following service request:

- http://localhost:8084/storageServices/s3/processControllers/processA/withObject/<json_object>/andDestination/<url>



Microservice: Process Controller Service

SERVICE OVERVIEW

The Process Controller Service (PCS) acts as the brain of the system and can be considered to be somewhat of an ESB (Enterprise Service Bus). This service provides the piping for the entire system as well as some logic for tying multiple services together and therefore does not strictly conform to the microservice constraints.

Source Code Repository

https://github.com/microdg/BETA-Process-Controller-Service.git

COMPOSITE TECHNOLOGIES

- Scala
- Spray (spray-can)
- Akka

CODE BASE OVERVIEW

Directory	**File**	**Description**
/	build.sbt	Standard SBT file listing all required dependencies and resolvers
src/main/scala/com/server	ProcessControllerServer.scala	HTTP integration layer exposes RESTful API endpoints and the server object is activated on boot
src/main/scala/com/services	ProcessControllerService.scala	Contains the main piping/process controlling functions
src/main/scala/com/support	CORSSupport.scala	A custom directive to enable Cross Origin Resource Sharing
	LoggingSupport.scala	Contains a function for logging service calls
	ProcessControllerHelper.scala	Contains many helper functions
	RouteHandlerService.scala	Contains functions which define specific routes
src/main/scala/com/models	RuntimeStatistics.scala	Defines the data model for a statistics object and provides two collections of objects for querying

API ENDPOINTS

Please note that all URIs are based on the scenario that the service is being run locally.

Endpoint Reference	api/processControllers/processA/locationType/<Location Type> /locationValue/<Location Name>
Description	Retrieve a composite data object comprised of information retrieved from several microservices (this service call does not collect runtime statistics)
URI:	Example: http://localhost:8083/api/processControllers/processA/locationType/town /locationValue/Skerries
HTTP verb	GET
Parameters	Location Type (Example: town) Data Type: Segment (String) Location Name (Example: Skerries) Data Type: Segment (String)
Pre Conditions	All other microservices are running
Post Conditions	The data is returned in JSON format for rendering and visualisation and the user is also presented with a hyperlink which they can use to download the JSON object to their local machine

Endpoint Reference	api/processControllers/processA/locationType/<Location Type> /locationValue/<Location Name>/withStats
Description	Retrieve a composite data object comprised of information retrieved from several microservices (this service call collects runtime statistics)
URI:	Example: http://localhost:8083/api/processControllers/processA/locationType/town /locationValue/Skerries withStats
HTTP verb	GET
Parameters	Location Type (Example: town) Data Type: Segment (String) Location Name (Example: Skerries) Data Type: Segment (String)
Pre Conditions	All other microservices are running
Post Conditions	The data is returned in JSON format for rendering and visualisation and the user is also presented with a hyperlink which they can use to download the JSON object to their local machine (a collection of runtime statistics is added to a queue)

Endpoint Reference	api/processControllers/processA/getStatistics
Description	Retrieve a JSON representation of a queue of statistic objects
URI:	http://localhost:8083/api/processControllers/processA/getStatistics
HTTP verb	GET
Parameters	None
Pre Conditions	None
Post Conditions	The data is returned in JSON format for rendering and visualisation

Endpoint Reference	api/processControllers/processA/getSampleStatistics
Description	Retrieve a JSON representation of a collection of pre-set statistic objects
URI:	http://localhost:8083/api/processControllers/processA/getSampleStatistics
HTTP verb	GET
Parameters	None
Pre Conditions	None
Post Conditions	The data is returned in JSON format for rendering and visualisation

SAMPLE RESPONSE

Process A with statistics

The following JSON object is a sample response from the microservice based on the following:

Endpoint: api/processControllers/processA/locationType/<Location Type>
/locationValue/<Location Name>
Parameter: Town
Parameter: Skerries


```
[
  {
    "file_location": "https:\/\/s3-eu-west-1.amazonaws.com\/microdg
test\/location_Skerries_accessed_2016-04-28_150.json",
    "b_name": "The Lifeboat Restaurant",
    "b_address": "Harbour Road Skerries Co Dublin",
    "b_phone": "+35318490109",
    "b_lat": "53.585465",
    "b_lng": "-6.1037089"
  }
]
```


Get Statistics

The following JSON object is a sample response from the microservice based on the following:

Endpoint: api/processControllers/processA/getStatistics
Parameter: None

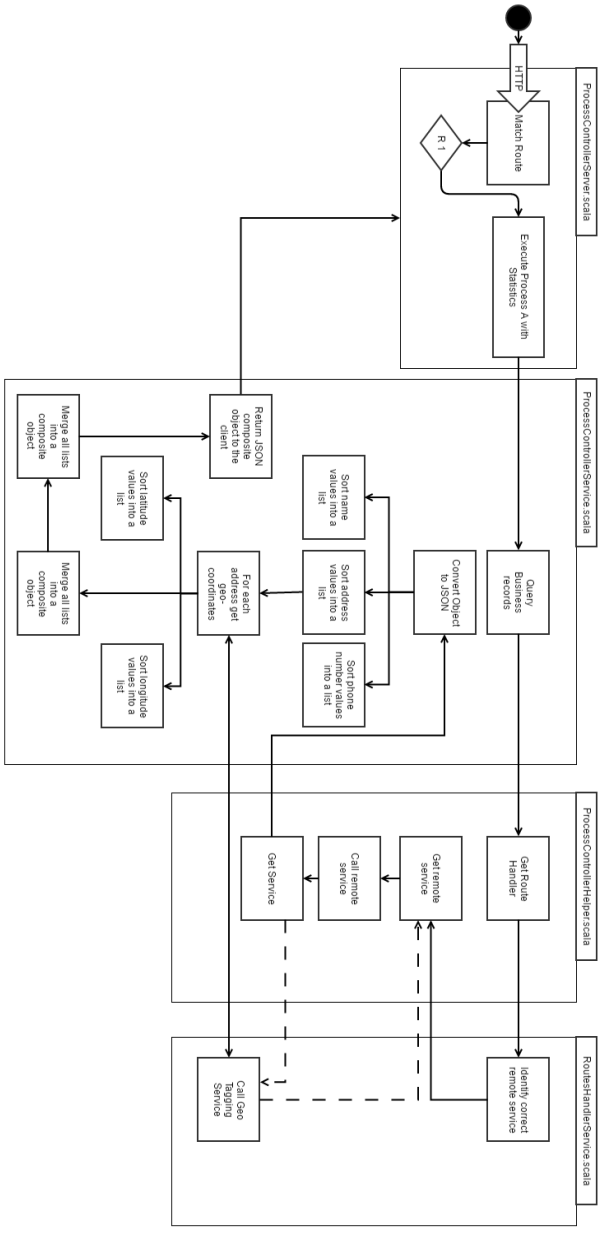

```
[
  {
    "time_stamp": "2016-04-28",
    "num_records_returned": 11,
    "runtime_pc": 2242,
    "runtime_bbds": 85,
    "runtime_gts": 2076,
    "runtime_ss": 35
  }
]
```

PROCESS FLOW DESCRIPTION

Process A with statistics

1. The PCS receives a HTTP GET request from the End User (EU) via the User Interface (UI).
2. The PCS parses the available routes and executes the request when the correct route has been identified/matched.
3. The appropriate function is actioned with the user parameters.
4. The PCS start time is recorded.
5. The BBDS start time is recorded.
6. The PCS gets the business records matching the query parameters from the BBDS service and stores it as an object.
7. The object is then converted to JSON.
8. The JSON object is iterated over three times. Each iteration uses the native Scala yield function to create a new collection for:
    1. Business Names
    2. Business Addresses
    3. Business Phone Numbers
9. The BBDS end time is recorded.
10. The GTS start time is recorded.
11. The Business Address collection is iterated over twice. Each iteration uses the native Scala yield function to create a new collection for:
    1. Latitude
    2. Longitude
12. Each of these new collections is iterated over again using the yield function. However, this time a function is applied to each value in order to strip away its quotation marks.
13. The GTS end time is recorded.
14. The SS start time is recorded.
15. A new file name is generated and stored as a string value.
16. This string is concatenated with a base URL string to form a complete URL. This URL will be later used as the S3 storage reference.
17. The size of the list is then recorded using the native length function.
18. A new locations collection is generated and populated with the location string generated in step 16.
19. The SS end time is recorded.
20. A new list composed of the six previously generated lists is created.
21. A Scala mapping function is applied to this new list. The purpose of this is to iterate over each list and extract the value at "i" in each list. That value is then used to create a new composite object. These composite objects are then used to build up the data sets collection.
22. The data set object is then converted to JSON.
23. A call is then made to the SS endpoint with the new data set JSON object and the generated filename as parameters. This results in the composite data object being stored in the remote S3 bucket.
24. The PCS end time is recorded.
25. The time difference between each services start and stop times are recorded in separate objects. The number of records returned by the process and the current date is also recorded.
26. The object at the head of the runtime statistics queue is popped.
27. A new object containing all of the runtime statistic values is then pushed into the queue.
28. The composite JSON object is returned to the HTTP integration layer.

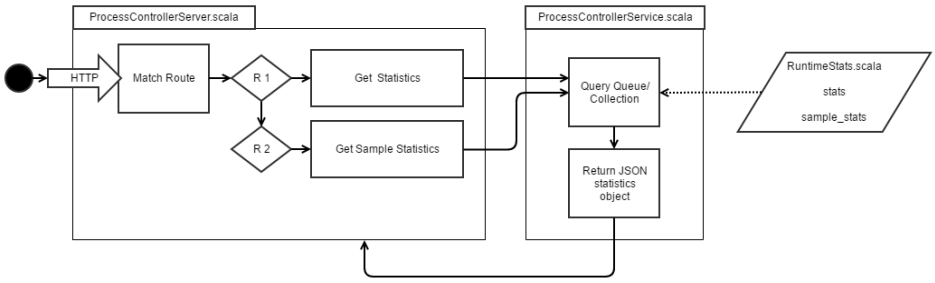
Process Flow Diagram: Process A with Statistics



Get Statistics

1. The PCS receives a HTTP GET request from the End User (EU) via the User Interface (UI).
2. The PCS parses the available routes and executes the request when the correct route has been identified/matched.
3. The statistics object is stored in a local variable and converted to JSON.
4. The JSON object is returned.

Process Flow Diagram: Get Statistics



FUNCTION/METHOD HIGHLIGHT

Process Controller

This is one of the main process controller functions and is responsible for retrieving and processing data from the Base Business Details Service as well as the Geo Tagging Service. The resulting data is then persisted to an Amazon S3 bucket via the AWS Storage Service.

```
      def processController_BG(queryType: String, queryValue: String): String = {

            LoggingSupport.logProgress("Get BBDS")

            val businessListMasterJson = ProcessControllerHelper.getRouteHandler(queryType,
queryValue)
            val businessListMasterJValue = parse(businessListMasterJson)

            val businessNames       =  for { JField("name", JString(name)) <-
businessListMasterJValue } yield name
            val businessAddresses   =  for { JField("address", JString(address)) <-
businessListMasterJValue } yield address
            val businessPhones      =  for { JField("phone", JString(phone)) <-
businessListMasterJValue } yield phone

            LoggingSupport.logProgress("Get GTS")

            def getGeoCords_lat(address: String): String = {
RouteHandlerService.processControllerGTS_lat(address) }
            def getGeoCords_lng(address: String): String = {
RouteHandlerService.processControllerGTS_lng(address) }
            def stripQuotes(x: String):           String = {x.replace("\"","")}

            val geoCordsListMasterJson_lat = for(address <- businessAddresses) yield
getGeoCords_lat(address)
            val geoCordsListMaster_lat     = for(lat <- geoCordsListMasterJson_lat) yield
stripQuotes(lat)

            val geoCordsListMasterJson_lng = for(address <- businessAddresses) yield
getGeoCords_lng(address)
            val geoCordsListMaster_lng     = for(lng <- geoCordsListMasterJson_lng) yield
stripQuotes(lng)

            LoggingSupport.logProgress("Combining Lists")

            case class DataSet(b_name: String, b_address: String, b_phone: String, b_lat:
String, b_lng: String)
            val min = List(businessNames, businessAddresses, businessPhones,
geoCordsListMaster_lat, geoCordsListMaster_lng).map(_.size).min
            val dataSets = (0 until min) map { i => DataSet(businessNames(i),
businessAddresses(i), businessPhones(i), geoCordsListMaster_lat(i),
geoCordsListMaster_lng(i)) }

            LoggingSupport.logProgress("Build & Return Json Object")

            val dataSetsJson = write(dataSets)

            return dataSetsJson
```

src/main/scala/com/services/ProcessControllerService.scala

Please see the process flow description above (process A with statistics)

Get Statistics

Here is a simple function to return a representation of a queue object in JSON format:

```
      //Function: Get runtime statistics from Queue
       def getRuntimeStatistics(): String = {
           val json = write(RuntimeStats.stats)
           return json
       }
```

src/main/scala/com/services/ProcessControllerService.scala

Please see the process flow description above (get statistics)

Route Handler

Below is an example of a route handler function. The purpose of this function is to act as internal piping for the application as well as logging service requests. All route handlers adhere to the same structure with the only variation being in the parameters they accept.

```
      //BBDS Query: Retrieve all records by Town Name
    def processControllerBBDS_town(queryValue: String, serviceName : String = "Base
Business Details (by town) Service", queryParam : String = "town") : String = {
        val content = ProcessControllerHelper.getRemoteService(serviceName, queryParam,
queryValue)
        LoggingSupport.checkReturnType(content)
        return content
    }
```

src/main/scala/com/services/RouteHandlerService.scala

1. The service name and query parameters are accepted.
2. The remote service is called and the response is stored in a variable called content.
3. The content object is returned.

Get Service

This is a simple helper function which takes a URL and makes a web service call. It then converts the response to a string.

```
        def getService(url: String) = scala.io.Source.fromURL(url).mkString
```

src/main/scala/com/services/ProcessControllerHelper.scala

Call Remote Service

This is a function which uses Scala's case matching functionality to identify the correct web service endpoint based on the input parameters. Once found, the function calls the "get service" function on the endpoint URL.

```
      def callRemoteService(queryParam: String, queryValue: String) : String = {
       val content = queryParam match {
                case "all" =>
getService("http://localhost:8081/baseBusinesses/all/details")
                case "town" =>
getService("http://localhost:8081/baseBbusinesses/all/details/towns/"+queryValue)
                case "county" =>
getService("http://localhost:8081/baseBusinesses/all/details/counties/"+queryValue)
                case "region" =>
getService("http://localhost:8081/baseBusinesses/all/details/regions/"+queryValue)
                case "gts" =>
getService("http://localhost:8082/geoTaggings/addresses/"+queryValue)
                case "gts_multi" =>
getService("http://localhost:8082/geoTaggings/addresses/"+queryValue)
                case "gts_lat" =>
getService("http://localhost:8082/geoTaggings/addresses/latitudes/"+queryValue)
                case "gts_lng" =>
getService("http://localhost:8082/geoTaggings/addresses/longitudes/"+queryValue)
                case _ => "No such Resource"
            }
     println(content)
     return content
    }
```

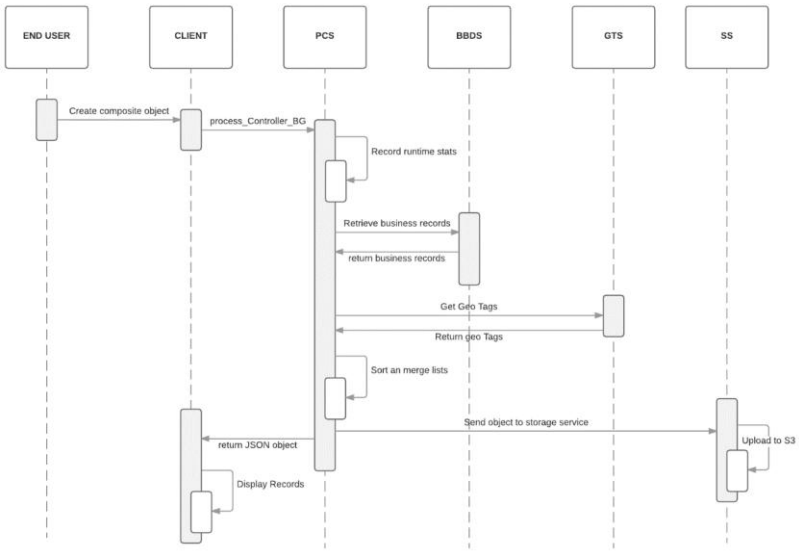
src/main/scala/com/services/RouteHandlerService.scala

1. Match the query parameter against a list of cases.
2. Execute the correct case statement with the query value.
3. Return the output.

CLIENT/SERVER SEQUENCE DIAGRAM

The following sequence diagram shows the interaction between the client and server for the following service request:

- http://localhost:8083/api/processControllers/processA/locationType/town /locationValue/Skerries



CODE COVERAGE

A suite of unit tests can be found under the src/test/scala/com/services directory. These tests employ the ScalaTest framework and are written using the "FlatSpec" trait. These unit tests cover the service and helper functions and ensure that they deliver the expected output. A detailed breakdown of these tests can be found in the Unit Tests (White Box) section (page 78).

Graphical User Interface (GUI)

The MicroDG client is a simple and easy-to-use user interface (UI) which was developed primarily using AngularJS and Bootstrap.

The structure of the user interface is as follows:

- **Landing page –** This presents the user with an overview of the MicroDG system as well as links to its individual components.
- **Service Gateway –** A consolidation of links to each constituent microservice.
- **Dashboards –** Interfaces for each of the following services:
  - Process Controller Service
  - Base Business Details Service
  - Geo Tagging Service
- **Analytics –** An analytics dashboard to view runtime statistics.

Each of the service interfaces have an AngularJS controller behind them which handles the interactions with the microservices and data retrieval via simple HTTP get requests.

SOURCE CODE REPOSITORY

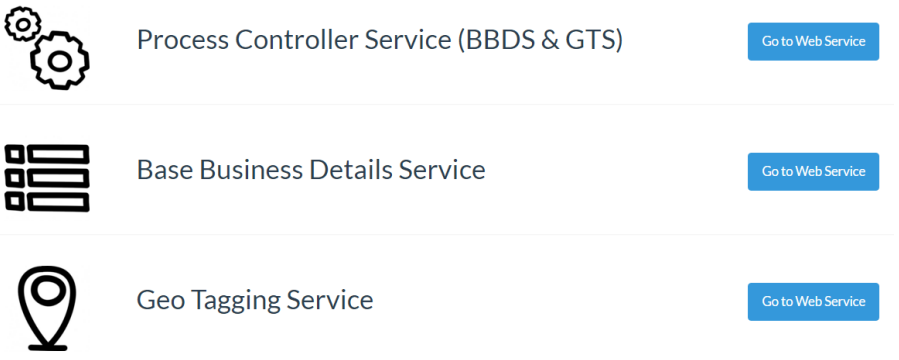
https://github.com/microdg/BETA-Client-UI.git

COMPOSITE TECHNOLOGIES

- AngularJS
- Bootstrap
- HTML5
- CSS3
- JavaScript
- Fusion Charts

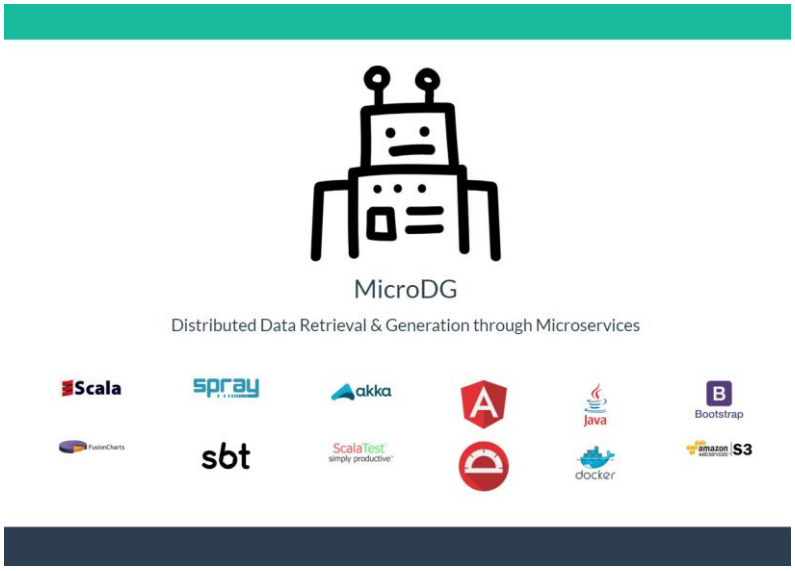
SERVICE GATEWAY

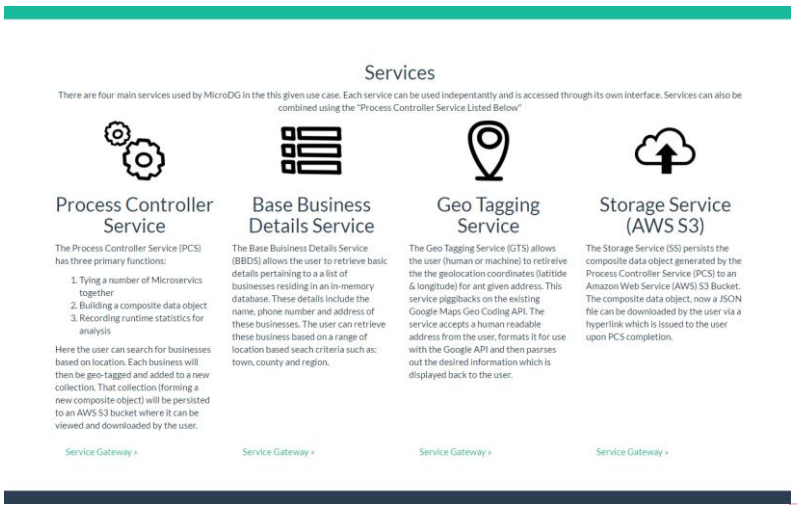
This simple interface provides the end user with a description of the accessible services and buttons which, when clicked, will navigate them to their chosen service.



LANDING/HOME PAGE

This is a simple landing page which presents the user with an overview of the MicroDG system as well as giving some brief explanations about each constituent service.



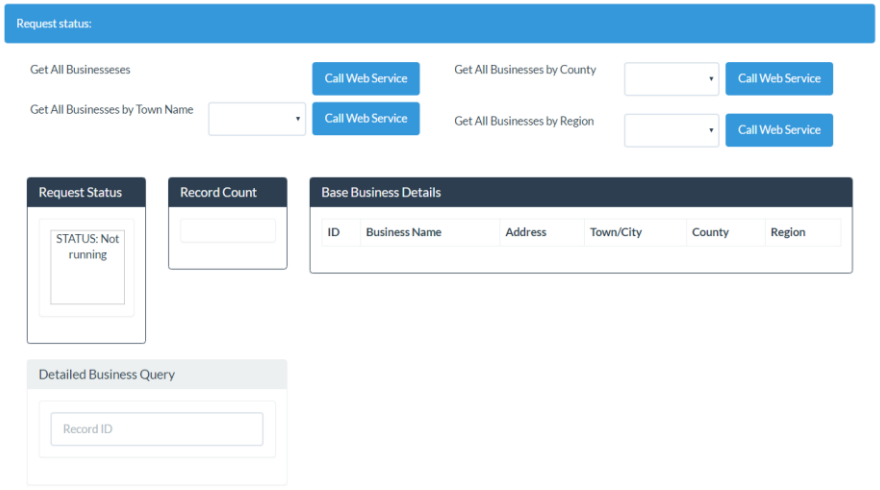


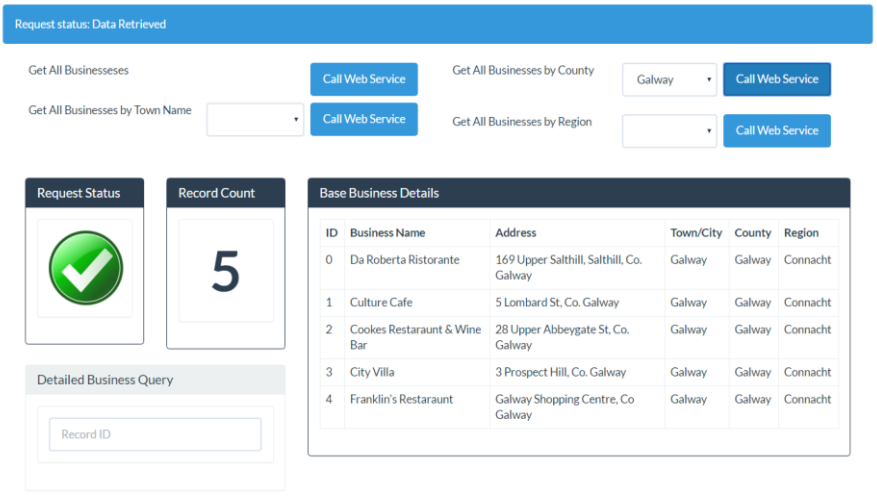
Commented [EOB1]: Typo in Base Business title

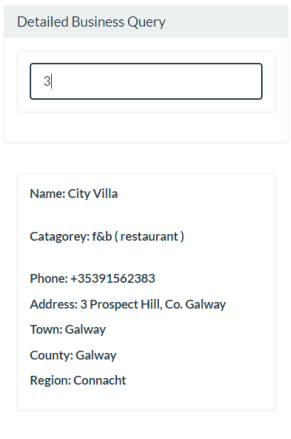
INTERFACE: BASE BUSINESS DETAILS SERVICE

This interface presents the user with a dashboard for displaying record information. It is comprised of:

1. A panel for displaying a request status image (green check if successful).
2. A panel for displaying the number of records returned.
3. A table for displaying all records returned.
4. An input form and panel for displaying a more detailed view of individual records based on their record ID.







Controller Functions

HTTP Request/Data Retrieval

```
$scope.getBusinessListByTown = function(townParam){
      $scope.loadStatus = "Retrieving Data...";
      $http.get("http://localhost:8081/baseBusinesses/all/details/towns/"+townParam)
                  .then(function (response) {
                    $scope.bbds_business_list = response.data;
                    $scope.count = response.length;
                    $scope.status_image = "http://www.clker.com/cliparts.jpg "
                    $scope.visibility_status = true;
                    $scope.loadStatus = "Data Retrieved";
      });
 };
```

app/controllers/BBDS.js

Dropdown Menu Options

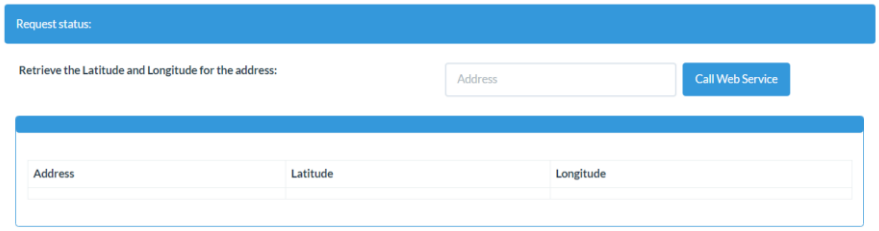
```
$scope.townOptions = {
  repeatSelect: null,
  availableOptions: [
    {id: 'Balbriggan', name: 'Balbriggan'},
    {id: 'Skerries', name: 'Skerries'},
    {id: 'Cork', name: 'Cork'},
    {id: 'Galway', name: 'Galway'},
    {id: 'Belfast', name: 'Belfast'}
  ],
 }
```

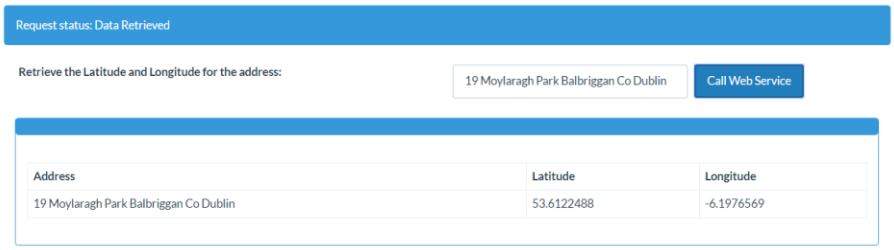
app/controllers/BBDS.js

INTERFACE: GEO TAGGING SERVICE

This interface presents the user with a dashboard for searching and displaying latitude and longitude coordinates based on an address inputted by the user. It is comprised of:

1. An input field and submit button.
2. A table for displaying the response.





Controller Functions

HTTP Request/Data Retrieval

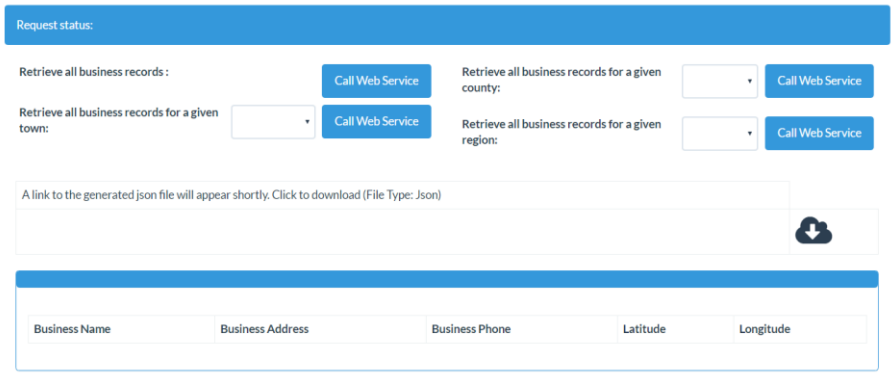
```
$scope.loadStatus = "";

$scope.getGeoTag = function(addressParam){
     $scope.loadStatus = "Retrieving Data...";
     $http.get("http://localhost:8082/geoTaggings/addresses/"+addressParam)
                  .then(function (response) {
                    $scope.g_cords_list = response.data;
                    $scope.query_address = addressParam;
                    $scope.loadStatus = "Data Retrieved";
     });
};
```

app/controllers/GTS.js

INTERFACE: PROCESS CONTROLLER SERVICE

This interface presents the user with a dashboard for retrieving data from several web services and combining them to form a new composite data object. The user is then presented with a hyperlink which, when they click it, will download a JSON file containing the composite data object from the MicroDG Amazon S3 bucket.

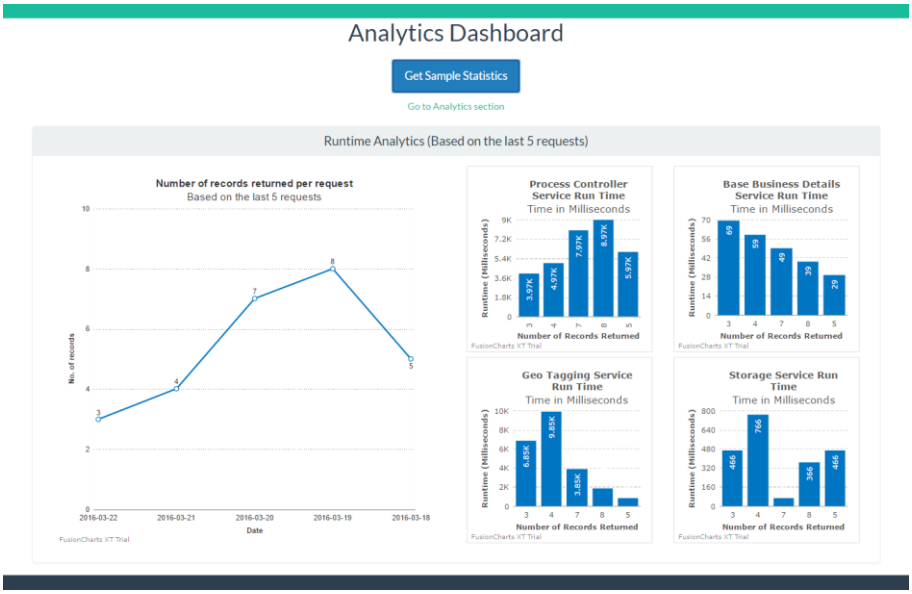


Business Name	Business Address	Business Phone	Latitude	Longitude
Brooks Hire	Unit BC5 M1 Business Park Courtlough Balbriggan Co. Dublin	+35318410436	53.56284059999999	-6.183885099999999
Harry Hire	3 Dublin Street Balbriggan Co. Dublin	+35318412417	53.6082901	-6.183329899999999
O' Rourke Plant Hire	51 Pinewood Green Lawn Balbriggan Co. Dublin	+35318415942	53.6009294	-6.1693146
Quads For Hire	Balrothery Balbriggan Co. Dublin	+353868723219	53.5921371	-6.1827698
Jack Doyles Bar	Bridge Street Balbriggan Co. Dublin	+35318413333	53.6098934	-6.1830655

INTERFACE: ANALYTICS DASHBOARD

This is a dashboard for displaying the runtime statistics which are recorded and stored each time the process controller service is run. The dashboard displays the number of records returned and length of time taken by each service to process the request.
The view is limited to the most recent five requests and updates each time the page is refreshed. For convenience, I have included a button which, when clicked, will load sample statistical data into the dashboard.

This dashboard uses the third party AngularJS charting library: Fusion Charts.



Library Dependencies

Provider	**Dependency Name**
com.typesafe.akka	akka-actor
io.spray	spray-routing
	spray-client
	spray-testkit
net.liftweb	Lift-json_2.10
com.typesafe.scala-logging	scala-logging-slf4j
ch.qos.logback	logback-classic
org.scalatest	Scalatest
org.mockito	mockito-all
com.amazonaws	aws-java-sdk

TESTING

Unit Testing (White Box)

OVERVIEW

The purpose of Unit Testing is to test that the individual functions/methods of a program are working correctly (i.e. producing the correct output given a specific input). Also known as white box testing, unit testing assumes low level, detailed, knowledge of the inner workings of a program but is only concerned with one unit at a time and not the overall system.

I have conducted unit tests for the following services:

- Base Business Details Service
- Geo Tagging Service
- Process Controller Service

A detailed description of the code coverage for each service is described below and includes relevant code snippets.

TEST FRAMEWORKS

All unit tests were created using the "FlatSpec" trait of the ScalaTest framework and are written entirely in the Scala language.

RUNNING TESTS

To run the following unit tests:

- Open a terminal window
- Navigate to the required directory
- Run the following command:
    - `sbt test`

BASE BUSINESS DETAILS SERVICE

ID	Description/Expected Result	PASS/FAIL
1	Calling the Business List function should return an object of type String	PASS
2	Calling the Business List function should have a length greater or equal to 0	PASS
3	Calling the Business List function should return a valid JSON array	PASS
4	Calling the Business List Count function should return an object of type Integer	PASS
5	Calling the Business List Count function should return an Integer greater or equal to 0	PASS
6	Calling the businessById function with parameter = 1 should return an object of type String	PASS
7	Calling the businessById function with parameter = 1 should have a length greater or equal to 0	PASS
8	Calling the businessById function with parameter = 1 should return a valid JSON array	PASS
9	Calling the businessesByTown function should return an object of type String	PASS
10	Calling the businessesByTown function should have a length greater or equal to 0	PASS
11	Calling the businessesByTown function should return a valid JSON array	PASS
12	Calling the businessesByTown function should have equal or less businesses than the master list	PASS
13	Calling the businessesByCounty function should return an object of type String	PASS
14	Calling the businessesByCounty function should have a length greater or equal to 0	PASS
15	Calling the businessesByCounty function should return a valid JSON array	PASS
16	Calling the businessesByCounty function should have equal or less businesses than the master list	PASS
17	Calling the businessesByRegion function should return an object of type String	PASS
18	Calling the businessesByRegion function should have a length greater or equal to 0	PASS
19	Calling the businessesByRegion function should return a valid JSON array	PASS
20	Calling the businessesByRegion function should have equal or less businesses than the master list	PASS
21	It should produce a welcome message	PASS

Sample Code Snippet

```
    //Return the full List of Businesses
    "Calling the Business List function" should "return an object of type String" in {
        assert(BBDRetrieval.businessList().isInstanceOf[String])
    }
    it should "have a length greater or equal to 0" in {
        assert(BBDRetrieval.businessList().length >= 0)
    }
    it should "should return a valid JSON array" in {
        assert(BBDRetrieval.businessList().startsWith("[") &&
BBDRetrieval.businessList().endsWith("]"))
    }
```

src/test/scala/com/BBDRetrievalSpec.scala

GEO TAGGING SERVICE

ID	Description/Expected Result	PASS/FAIL
1	Calling the GET Latitude function with the sample Google Geocoding response should return the value 53.6122488	PASS
2	Calling the GET Longitude function with the sample Google Geocoding response should return the value -6.1976569	PASS
3	Calling the URL Constructor function with the sample address: 19 Moylaragh Park Balbriggan Co Dublin should return the valid URL: https://maps.googleapis.com/maps/api/geocode/json?address=19+Moylaragh+Park+Balbriggan+Co+Dublin&key=AIzaSyCW6S4V0IgkgoXwsgZUtdTTNRzM2zPqe9E	PASS
4	Calling the GET Geo Tags (Lat & Lng) function should return an object of type String	PASS
5	Calling the GET Geo Tags (Lat & Lng) function should have a length greater or equal to 0	PASS
6	Calling the GET Geo Tags (Lat & Lng) function should return a valid JSON array	PASS
7	Given the address: 19 Moylaragh Park Balbriggan Co Dublin, the GET Geo Tag (Lat & Lng) function should return the coordinates lat 53.6122488, lng -6.1976569	PASS
8	Calling the GET Geo Tags (Lng) function should return an object of type String	PASS
9	Calling the GET Geo Tags (Lat) function should return an object of type String	PASS
10	Calling the GET Geo Tags (Lat) function should have a length greater or equal to 0	PASS
11	Given the address: 19 Moylaragh Park Balbriggan Co Dublin, the GET Geo Tag (Lat) function should return the coordinates lat 53.6122488	PASS
12	Calling the GET Geo Tags (Lng) function should have a length greater or equal to 0	PASS
13	Given the address: 19 Moylaragh Park Balbriggan Co Dublin, the GET Geo Tag (Lng) function should return the coordinates lng -6.1976569	PASS

Sample Code Snippet

```
    val testAddress = "19 Moylaragh Park Balbriggan Co Dublin"
    val expectedLat = "53.6122488"
    val expectedLong = "-6.1976569"


   "Calling the Get Geo Tags (Lat) function" should "return an object of type String" in {
        assert(GTRetrievalService.getGeoLats_single(testAddress).isInstanceOf[String])
    }
    it should "have a length greater or equal to 0" in {
        assert(GTRetrievalService.getGeoLats_single(testAddress).length >= 0)
    }

    s"Given the address: $testAddress, the Get Geo Tag (Lat) function" should s"return the
coordinates lat $expectedLat" in {
        assert(GTRetrievalService.getGeoLats_single(testAddress).contains(expectedLat))
    }
```

src/test/scala/com/GTRetrievalSpec.scala

PROCESS CONTROLLER SERVICE

ID	Description/Expected Result	PASS/FAIL
1	Calling the recordCount function on the test Data Set should produce the integer value 3	PASS
2	Passing the parameter 115A Sarsfield Park Lucan Co Dublin to the encode URL function should return 115A+Sarsfield+Park+Lucan+Co+Dublin	PASS
3	Calling the Remote Service switch function with the parameters gts and 115A Sarsfield Park Lucan Co Dublin should return content matching the expected JSON provided	PASS
4	Calling the GET Service function with the parameters: http://localhost:8082/geoTaggings/addresses/115A+Sarsfield+Park+Lucan+Co+Dublin should return content matching the expected JSON provided	PASS
5	Calling the GET Remote Service function with the parameters: Geo Tagging Service (retrieve lat & lng), gts and 115A Sarsfield Park Lucan Co Dublin should return content matching the expected JSON provided	PASS
6	Calling the TimeStamp function should return the correct time in milliseconds	PASS
7	Calling the Time Difference function should return an integer value of zero	PASS

Sample Code Snippet

```
s"TEST-3: Passing the parameter $sampleParam4 to the encode url function" should s"return
$encodedUrl" in {
        assert(ProcessControllerHelper.encodeUrl(sampleParam4) == encodedUrl)
    }

    s"TEST-4: Calling the Remote Service switch function with the parameters $sampleParam3
and $sampleParam4" should "retrun content matching the expected JSON provided" in {
        assert(ProcessControllerHelper.callRemoteService(sampleParam3, encodedUrl) ==
expectedJson_gts)
    }

    s" TEST-5: Calling the Get Service function with the parameters: $sampleParam5" should
"retrun content matching the expected JSON provided" in {
        assert(ProcessControllerHelper.getService(sampleParam5) == expectedJson_gts)
    }
```

src/test/scala/com/ProcessComtrollerHelperSpec.scala

End to End Testing (Black Box)

OVERVIEW

The purpose of End to End (e2e) testing, also known as functional or integration testing, is to test the user/data flow from start to finish. It is used to replicate work flows or processes to ensure that each component in the system is working correctly both by themselves and in collaboration with others.

End to End testing is considered to be "Black Box" testing because the tester does not need to understand the inner workings of the system. Instead, these tests are only concerned with the end result, so that when given a specific input, the system produces the expected output. There is no consideration given as to how the system achieves it, only that the system achieves it.

I have conducted e2e tests for the following services:

- Base Business Details Service
- Geo Tagging Service
- Process Controller Service

A detailed description of the code coverage for each service is described below and includes relevant code snippets.

TEST FRAMEWORKS

All e2e tests were developed using the Protractor framework for AngularJS. This framework provided the ability to replicate user interface interaction via browser automation and test a variety of user paths through the system.

RUNNING TESTS

To run the following e2e tests:

- Open a terminal window
- Navigate to the required directory
- Start the web driver server by running the following command:
    - `webdriver-manager start`
- Open a second terminal window
- Navigate to the required test directory
- Run the e2e tests using the following command:
    - `protractor conf.js`

BASE BUSINESS DETAILS SERVICE

ID	Description/Expected Result	PASS/FAIL
1	Should have a title	PASS
2	Should return the correct business details from a search based on the "ALL" parameter	PASS
3	Should return the correct business details from a search based on the "TOWN" parameter	PASS
4	Should return the correct business details from a search based on the "COUNTY" parameter	PASS
5	Should return the correct business details from a search based on the "REGION" parameter	PASS

Sample Code Snippet

```
    it('should return the correct business details from a search based on the "ALL"
parameter', function() {
      element(by.id('allSearch')).click();
      var count = element(by.binding('bbds_business_list.length'));

      var name1 = element(by.exactRepeater("record in
bbds_business_list").row(0).column("record.name"));
      var address1 = element(by.exactRepeater("record in
bbds_business_list").row(0).column("record.address"));
      var town1 = element(by.exactRepeater("record in
bbds_business_list").row(0).column("record.town"));
      var county1 = element(by.exactRepeater("record in
bbds_business_list").row(0).column("record.county"));
      var region1 = element(by.exactRepeater("record in
bbds_business_list").row(0).column("record.region"));

      expect(count.getText()).toEqual("38");

      expect(name1.getText()).toEqual("Brooks Hire");
      expect(address1.getText()).toEqual("Unit BC5 M1 Business Park Courtlough Balbriggan
Co. Dublin");
      expect(town1.getText()).toEqual("Balbriggan");
      expect(county1.getText()).toEqual("Dublin");
      expect(region1.getText()).toEqual("Leinster");

});
```

app/test/bbds-e2e-spec.js

GEO TAGGING SERVICE

ID	Description/Expected Result	PASS/FAIL
1	Should have a title	PASS
2	Should return the correct lat & lng values	PASS

Sample Code Snippet

```
it('should return the correct lat & ln values', function() {
      element(by.model('addressParam')).sendKeys('115A Sarsfield Park Lucan Co Dublin');

      element(by.id('callWS')).click();

      expect(element(by.binding('query_address')).getText()).
        toEqual('115A Sarsfield Park Lucan Co Dublin');

      expect(element(by.binding('g_cords_list[0].lat')).getText()).
        toEqual('53.35870269999999');

      expect(element(by.binding('g_cords_list[0].lng')).getText()).
        toEqual('-6.4438657');
  });
```

app/test/gts-e2e-spec.js

PROCESS CONTROLLER SERVICE

ID	Description/Expected Result	PASS/FAIL
1	Should have a title	PASS
2	Should return the correct business details from a search based on the "ALL" parameter	PASS
3	Should return the correct business details from a search based on the "TOWN" parameter	PASS
4	Should return the correct business details from a search based on the "COUNTY" parameter	PASS
5	Should return the correct business details from a search based on the "REGION" parameter	PASS

Sample Code Snippet

```
it('should return the correct business details based on the "ALL" parameter', function() {
      element(by.id('allSearch')).click();

      var name1 = element(by.exactRepeater("record in
obj").row(0).column("record.b_name"));
      var address1 = element(by.exactRepeater("record in
obj").row(0).column("record.b_addres"));
      var phone1 = element(by.exactRepeater("record in
obj").row(0).column("record.b_phone"));
      var lat1 = element(by.exactRepeater("record in obj").row(0).column("record.b_lat"));
      var lng1 = element(by.exactRepeater("record in obj").row(0).column("record.b_lng"));

      expect(name1.getText()).toEqual("Brooks Hire");
      expect(address1.getText()).toEqual("Unit BC5 M1 Business Park Courtlough Balbriggan
Co. Dublin");
      expect(phone1.getText()).toEqual("+35318410436");
      expect(lat1.getText()).toEqual("53.56284059999999");
      expect(lng1.getText()).toEqual("-6.183885099999999");

});
```

app/test/pcs-e2e-spec.js

User Acceptance Testing

OVERVIEW

The purpose of User Acceptance Testing (UAT) is to determine whether or not the system meets the overall project requirements at the end of development. This form of testing involves actual end users manually using the system to replicate specific scenarios.

When testing each scenario, the end user will be presented with a form which they will fill out to determine whether or not the system passes or fails the set requirements. In addition to this, the user will also be asked to grade certain features and provide some general feedback on the system.

I have divided the UAT into two groups:

1. The first consists solely of the project sponsor whose general requirements helped to guide the development of this system.
2. The second will consist of a diverse group of users. Not all of the users will have an IT or Business/Marketing background and not all of these users will have a need to use such a system in their daily lives.

SCENARIOS

Below is a table of the scenarios which users will be asked to pass or fail:

ID	Scenario	PASS/FAIL
1	Navigate easily from the home page to each composite service dashboard	
2	Retrieve all business listings from the BBDS service	
3	Retrieve business listings from the BBDS service based on TOWN name	
4	Retrieve business listings from the BBDS service based on COUNTY name	
5	Retrieve business listings from the BBDS service based on REGION name	
6	View a detailed view of a business record given an ID	
7	Retrieve the latitude and longitude coordinates for an address via the GTS	
8	Generate a composite business object of all businesses using the PCS	
9	Generate a composite business object of all businesses, based on TOWN name, using the PCS	
10	Generate a composite business object of all businesses, based on COUNTY name, using the PCS	
11	Generate a composite business object of all businesses, based on REGION name, using the PCS	
12	Download a composite business object from the cloud to the local machine	
13	View runtime statistics on an analytics dashboard	
14	Generate and view sample runtime statistics on an analytics dashboard	

USER EXPERIENCE

Users will be asked to grade their user experience between 1 (low) and 5 (high) under the following headings:

• Ease of use • Speed • Usefulness	• Response time • Look and feel • Reusability

GROUP ONE: PROJECT SPONSOR

Name:	Emma O'Brien	**Date:**	05/05/2016
Occupation:	Marketing and Sales Coordinator	**IT Literacy:**	Advanced

ID	Scenario	PASS/FAIL
1	Navigate easily from the home page to each composite service dashboard	Pass
2	Retrieve all business listings from the BBDS	Pass
3	Retrieve business listings from the BBDS based on TOWN name	Pass
4	Retrieve business listings from the BBDS based on COUNTY name	Pass
5	Retrieve business listings from the BBDS based on REGION name	Pass
6	View a detailed view of a business record given an ID	Pass
7	Retrieve the latitude and longitude coordinates for an address via the GTS	Pass
8	Generate a composite business object of all businesses using the PCS	Pass
9	Generate a composite business object of all businesses, based on TOWN name, using the PCS	Pass
10	Generate a composite business object of all businesses, based on COUNTY name, using the PCS	Pass
11	Generate a composite business object of all businesses, based on REGION name, using the PCS	Pass
12	Download a composite business object from the cloud to the local machine	Pass
13	View runtime statistics on an analytics dashboard	Pass
14	Generate and view sample runtime statistics on an analytics dashboard	Pass

User Experience
Please rate between 1 – 5 (1 being low and 5 being high)

Ease of use	____5____	**Response time**	____4____
Speed	____4____	**Look and feel**	____5____
Usefulness	____5____	**Reusability**	____5____

General Feedback

Very clean look and feel to the website.
Very easy to navigate.
Would love to see this being commercialised.

Future Improvements

Replace "call web service" with a simple call to action (for example "Go").
Prior to detailed base business details service query, do not show the blank fields.
Perhaps expand the database to include countries outside of Ireland.

GROUP TWO: ASSORTED USER GROUP

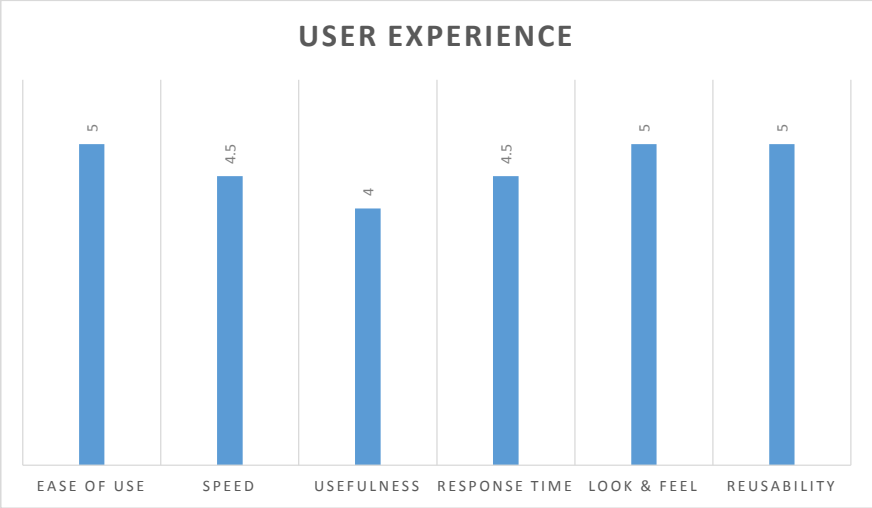
The group of non-sponsor users consisted of software testers, teachers, students, sales managers and administrative workers.

I have compiled their feedback regarding user experience in the chart below with the figures listed being the average score amongst the group.

Each member's full UAT feedback forms can be viewed in Appendix F.

User Experience Chart

The following chart represents the feedback regarding the user experience. Each category is graded between one and five, with one being poor and five being excellent.



DEPLOYMENT

One of the key benefits to using a microservice architecture is that each composite service can be deployed separately. This can be very important for a number of reasons.

Firstly, when looking to scale a system out either vertically or horizontally, not all services will require the same level of power or availability. By having each service separated from the others, strategies can be implemented so that only the service(s) that require scaling receive it. This results in efficient resource management and reduced overheads. An example of how this would affect the MicroDG system is as follows:
A call to the process controller function which retrieves one hundred records will trigger:

	PCS	BBDS	GTS	SS
Service Calls	1	1	100	1

As you can see above, most services are being called just once, however the geo tagging service is being called for the number of addresses returned by the query to the base business details service (in this case one hundred times). Therefore in the above scenario, only the GTS service should be scaled out.

Secondly, a company may have specific data requirements which restrict the geographical locations they are permitted to use for data storage or processing. In some cases however, these requirements do not apply to all of their services. It may therefore be beneficial to deploy specific services across different global regions rather than deploying the entire system in one region. This ensures that the system will not be majorly impacted by the requirements of a single restrictive service.

Deployment Strategy

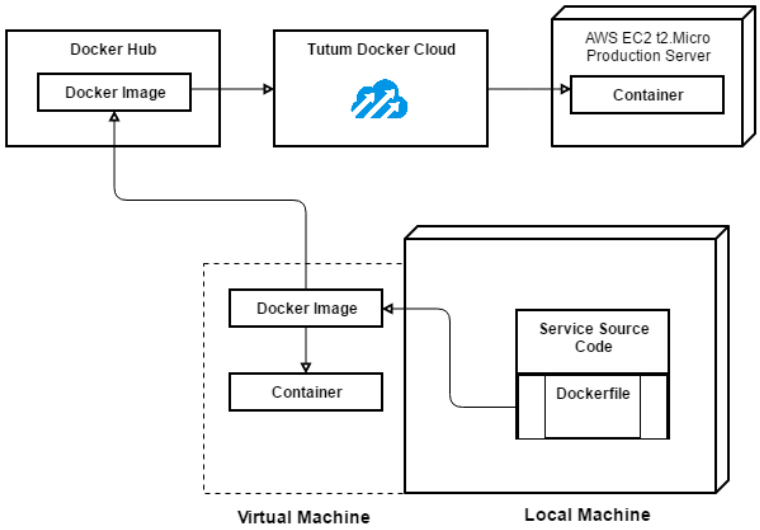
The deployment strategy for MicroDG is to deploy each service separately using a combination of Docker, Tutum and AWS EC2.

The first step is the “Dockerise” each service. This involves creating a Docker image which is essentially the services source code and runtime environment. This image is then pushed to the MicroDG Docker Hub repository and is then later used to spin up service containers.

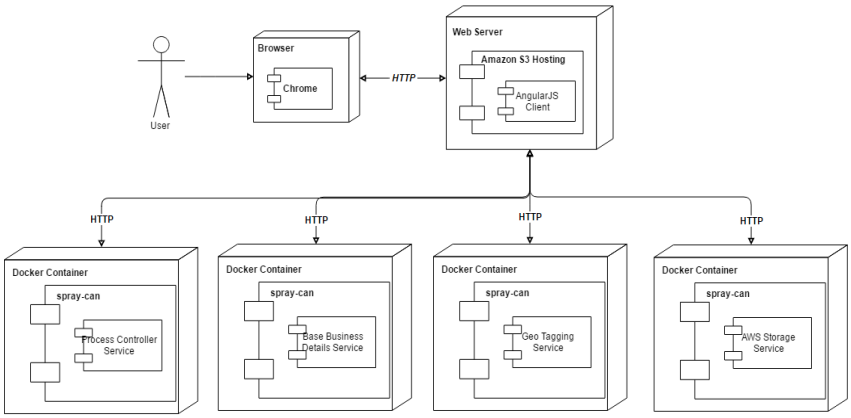
The second step is to use the Tutum Docker Cloud which provides a simple way to interface between Docker Hub and an AWS EC2 instance.

The third step is then to mount the Docker container (which includes the service source code and environment) to the EC2 t2.Micro server(s).

DOCKER DEPLOYMENT OVERVIEW



DEPLOYMENT DIAGRAM



CONCLUSIONS

The development of the MicroDG system has been an incredibly challenging but rewarding experience for me. The technology stack I used was modern and industry leading, however this meant that I had very little knowledge prior to development.

It would have been a much easier option for me to develop the system using a framework which had been covered throughout the course of my degree such as ASP.NET or Ruby on Rails. However, I felt that taking this option would be counterproductive towards my learning and career goals.

As a mature student, my goal has always been to earn a first class honours degree and secure a job in one of the best cloud-based tech companies in the world. For those reasons, I felt that the project idea I chose and the technology I chose to implement it must reflect what employers are looking for right now and not what they were looking for when I started my degree.

One unexpected development which arose from this project was my new interest and appreciation for cyber security. I unfortunately fell victim to a cyber attack whereby my Amazon Web Services account was hacked and the attackers ran up a bill of over seven thousand euro in the space of 24 hours. Luckily, the issue has now been rectified but it has inspired a new interest in the field of security which I had not previously explored.

The feedback received from the project sponsor was very positive and indicates the success of this endeavour, as do the positive reviews I have received from my peers.

I feel that I have learned a great deal from developing this system as my final year project and feel that it has equipped me to enter the workforce and bring value to any company I join.

Further Developments

Functionally, MicroDG does what it set out to do – it provides the user with a way to generate a composite data object based on information sourced from several distributed services.

The particular use case here was automating the task of a marketing department in terms of field sales lead generation. However, MicroDG does not have to be limited to just tagging business addresses with their geo coordinates. The true potential of this system is in fact limitless. The microservices architecture used by this system means that, with the independent development of services and some slight modification to a process controller, the system's functionality could expand exponentially.

The sales-lead data market has always been an expensive business, with the likes of call centres and similar service providers spending hundreds of thousands of euro each year for raw customer data. The cost of this data is reflective of how much the data can yield in terms of sales revenues but also how much it costs to manually gather and process the data. MicroDG could potentially reduce that overhead significantly and offer a much more affordable service to customers on a subscription basis.

In terms of specific changes to the system in its current form, I would implement the following:

1. Persistent storage of query parameters and resource links. This would introduce an intermediary step between when the user requests a new composite data object to be generated and when they receive the download link. There may be many instances where users are requesting an object to be created with the same query parameters as other users. The proposed modification to the system would allow for a query to be made against a database to see if those query parameters had been used previously. If they had, the system should return the link to the pre-existing S3 object rather than generate a new one.
2. Inclusion of Eircode data. The original proposal for this system included the ability to tag a business address with its corresponding Eircode details. This would have been simple to implement using the "Auto Address" web service. I had made contact with the Auto Address marketing manager requesting a free developer API key, however the company were not happy to issue one.
3. Finally, I would implement a service which would allow the user to choose the format in which they want to receive the composite data at the end of the process. This could, for example, be a JSON, XML or an Excel spreadsheet.

BIBLIOGRAPHY

Akka.io. “Akka”. N.p., 2016. Web. 2 Feb. 2016.

Docker. “What Is Docker?”. N.p., 2015. Web. 4 Feb. 2016.

Lewis, James, and Fowler, Martin. “Microservices”. *martinfowler.com*. N.p., 2014. Web. 31 Jan. 2016.

Liddle, Jim. “Amazon Found Every 100Ms Of Latency Cost Them 1% In Sales. Gigaspaces XAP Blog – The In-Memory Computing Platform”. *Blog.gigaspaces.com*. N.p., 2016. Web. 4 Feb. 2016.

Scala-lang.org. “The Scala Programming Language”. N.p., 2016. Web. 2 Feb. 2016.

Spray.io. N.p., 2016. Web. 2 Feb. 2016.

Raya, Jose. “CORS Directive For Spray”. *Gist*. N.p., 2016. Web. 27 Apr. 2016.

APPENDIX

Appendix A: Project Proposal

Objectives

I propose building a scalable system of independent microservices which can be used in conjunction with each other to dynamically generate marketing data for third party users.

The services will allow the user to build an object comprising of data pertaining to a list of businesses and their addresses, phone numbers, categories, geo locations and Eircodes. The nature of this system, however, will allow for additional services to be added in the future should the need arise.

The services will be accessible both individually and as a combination via a web browser and, upon completion, will produce a file (most likely a Google Spreadsheet) which the user can download via a hyperlink.

The system is targeted at marketing departments that require bulk generation of field sales information.

Service Overview

Base Business Details Service	A service which retrieves a list of businesses and their base details from a remote database.
Geo Tagging Service	A service which retrieves the latitude and longitude coordinates for each address passed to it.
Eircode Tagging Service	A service which retrieves the postal code (Eircode) for each address passed to it.
Google Drive Interaction Service	A service which connects to an instance of Google Drive via a service account and creates and populates a Google Spreadsheet with values contained within a JSON object.
API Gateway Service	A process controller which acts as a proxy between the UI and the individual web services.

Background

I have three key motivations for building this proposed system:

1. I have spoken to Emma O'Brien, a key member of the Marketing team at Mcor. One of her daily tasks is compiling information for field sales agents. The typical process includes gathering data from their internal lead generation process and then manually gathering supporting data, for example geo coordinates for each lead's address. This task can be time-consuming based on the high volumes of leads that are generated on a weekly basis. This proposed system was of interest to Ms O'Brien because she felt it could potentially reduce the time required to complete the above task. Ms O'Brien has offered to assist in the requirements elicitation process which will help to guide the final system design.
2. This system will allow me to demonstrate some of the knowledge I have gained from my final year specialisation. Each service can be considered to be SAAS and I will be utilising certain

PAAS offerings such as database as a service. Also, as some services (such as the Geo Tagging Service) will experience higher levels of activity, it will allow me to practise scalability strategies for individual services rather than on the system as a whole.

3. What is most important to me about this proposed system is not the concept behind the application but the technology stack I propose using to deliver it. I am very fortunate to have signed a permanent contract with a fantastic company, Workday, Inc., as an Associate Software Development Engineer, following a nine-month work placement during the third year of my studies. The role I will be going into this coming June is one which uses some, if not all, of my proposed technology stack for this project. Also, by the time I return to Workday, my team will be in the process of migrating the product I worked on to a microservices architecture pattern, so any experience gained now will be of a huge advantage to me. The concept of my final year project has no relevance to the product offerings of Workday, but I see this project as an opportunity to become better versed in the tools and technology which I will be using in my future career.

While conducting market research for this project proposal, I was made aware of several premium lead generation offerings available such as Data Ireland which offers end products such as the one I am proposing. However, I feel that there are enough variants to justify me choosing this domain for my project.

Technical Approach

Development

I propose using both Scala and Ruby as the primary development languages for this project along with the Spray and Akka toolkits to provide a REST/HTTP integration layer for each service.

In terms of development style, I aim to adopt a test-driven development method which will ensure that, at the end of each sprint, the application will have full test coverage (functional, system, end-to-end, etc.) for all new features.

Architectural Design/Pattern

In terms of architecture, I will be adopting a microservices design pattern which will allow me to deliver the individual constituent components of the application, fully functional and independent of one another.

Project Management

I will be adopting an agile approach to this project as it is the methodology I have become most familiar with over the past year. I will identify user stories and then break them down into deliverable epics, tasks and subtasks.

I will require a large amount of technical research due to the technologies I have chosen and for that reason I will be using the time prior to January 2016 for "Spike" related tasks.

Starting from January 2016, I will commence a series of development sprints, each two weeks in duration and with the aim of delivering a new service at the end of each sprint.

At the end of sprint one, an MVP (Minimum Viable Product) should be delivered. This will comprise of one working service partnered with a user interface. The MVP should be available for the Mid-Point Presentation in February. Each sprint after that will add functionality to the system until the finished product is delivered.

Special Resources Required

Software	There are no special software resources required for this project.
Hardware	Remote servers are required for the deployment of this application. Please see the Technical Details section below for details on the cloud platforms I intend to use. Multiple devices such as desktop/mobile devices may be required for UX testing and these are currently available to me through personal ownership.
Documentation	Extensive documentation exists for the technology stack I intend to use.

Technical Details

I plan on implementing an actor (Akka) based microservices architecture pattern to help build an elastic and scalable distributed application.

Proposed Tools/Technology

Development

- Spray
- Play
- Scala
- Akka
- Ruby
- Docker
- Bootstrap
- AngularJS
- MySQL
- HTML5
- CSS3

Testing

- Selenium
- ScalaTest framework

Continuous integration and build tools

- Jenkins

APIs/Services Consumed

Amongst others, I intend to use the following APIs:

Google Maps Geocoding API - This API allows for the conversion of human readable addresses (such as “100 Main Street, Balbriggan, County Dublin”) into their Latitude and Longitude equivalent. As this API is designed for the conversion of addresses from static data sources as opposed to dynamic real time input, it is the most appropriate for the intended purpose.

Google Drive API - Amongst other services, this API allows for the uploading and downloading of files to a user's Google Drive account, which is an essential feature of this application.

Google Sheets API - This API provides an interface to read and modify content within Google Spreadsheets.

Auto Address API - This API allows for the retrieval of an Eircode based on a valid human readable address.

Deployment
I plan on deploying this application on the following cloud platforms:

- AWS
- Heroku

The reason for my choice is that they both support scalable applications built using Scala and Ruby and both provide "Free Tiers", as a paid platform is not currently an option for me.

Evaluation
Like any application, testing is essential to the delivery of a successful application. As mentioned above, I propose adapting the test-driven development method whereby I develop functional tests whilst developing new application features. This approach will provide me with fully functional test coverage.

For End to End (e2e) testing, I will utilise Selenium software to replicate user interaction from start to finish along a variety of user/data flow paths.

Finally, I will conduct acceptance testing with physical end users to measure how intuitive the UX (user experience) is.

David Brady 02/08/2015

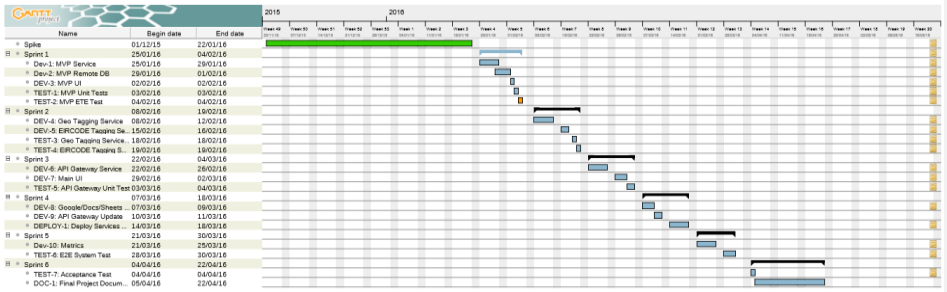
Appendix B: Project Plan

Name	Begin date	End date
Spike *The purpose of this spike is to research the proposed technolgy stack and architecture patterns.* *- Microservices* *- REST* *- Spray* *- Akka* *- Scala* *- Ruby* *- APIs* *- AWS* *- Heroku* *- Docker*	01/12/15	22/01/16
Sprint 1 *Sprit 1* *Purpose f this sprint is to create the MVP which will comprise of the Base Business Details Service, A relational DB and a UI*	25/01/16	04/02/16
Dev-1: MVP Service *Build a service using spray and scala which returns a json object composed of a list of Base Business Details:* *- Name* *- Adddress* *- Catagorey* *- Sub Catagorey* *- Phone* *- Town/City* *- County* *- Region* *It is acceptable for the prototype to read the json object directly from the source code of from a static file*	25/01/16	29/01/16
Dev-2: MVP Remote DB *Create an populate a remote Database (suggested AWS My SQL). Then Modify the Base Business Details Service to read from this Database*	29/01/16	01/02/16
DEV-3: MVP UI *Create a simple AngularJS User Interface which retrieves and Displays the Base Business Details via a HTTP Get Request*	02/02/16	02/02/16
TEST-1: MVP Unit Tests *Create a Suite of Unit Tests using the ScalaTest Framework and the FlatSpec pattern.*	03/02/16	03/02/16
TEST-2: MVP ETE Test *Create End to End Test for the service if time allows it*	04/02/16	04/02/16
Sprint 2	08/02/16	19/02/16

Name	Begin date	End date
DEV-4: Geo Tagging Service *Create a new service for the retrieval of latitude and longitude coordinates based on a given address.* *Notes:* *The response should be in the form of a Json object.* *Google Geo Tagging Service API should be used*	08/02/16	12/02/16
DEV:-5: EIRCODE Tagging Service *Create a service which retrieves the Eircode for a given address.* *Notes:* *The response should be in the form of a JSON string* *The Auto Address API should be used*	15/02/16	16/02/16
TEST-3: Geo Tagging Service Unit Tests *Create a a series of unit tests for the Geo Tagging Service using the ScalaTest Framework and the FlatSpec Pattern*	18/02/16	18/02/16
TEST-4: EIRCODE Tagging Service *Create a a series of unit tests for the Eircode Tagging Service using the ScalaTest Framework and the FlatSpec Pattern*	19/02/16	19/02/16
Sprint 3	22/02/16	04/03/16
DEV-6: API Gateway Service *Create a proxy service which handles HTTP Requests between the UI and the individual web services* *Notes:* *This service should provide a process controller service which links a number of web services together.*	22/02/16	26/02/16
DEV-7: Main UI *Create the Main UI for interacting with both the individual services and the API Gateway.* *NOTES:* *The UI should comprise of:* *- A landing page which displays basic information about teh application.* *- A dashboard to dispaly certain statistcal data* *- A page with a series of imput forms to the services* *- A page with API documentation*	29/02/16	02/03/16
TEST-5: API Gateway Unit Test *A series of Scalatests to ensure that the API Gateway is working correctly*	03/03/16	04/03/16
Sprint 4	07/03/16	18/03/16

Name	Begin date	End date
DEV-8: Google/Docs/Sheets interation	07/03/16	09/03/16
A service written in Ruby that will connect to an instance of Google Docs and write the contents of a JSON object (containing the composed Business Details) to a spreadsheet		
NOTES: *The service should name the spreadsheet with a date stamp and return the url as a JSON string*		
DEV-9: API Gateway Update	10/03/16	11/03/16
DEPLOY-1: Deploy Services & UI	14/03/16	18/03/16
Deploy each service on either AWS or Heroku		
Sprint 5	21/03/16	30/03/16
Dev-10: Metrics	21/03/16	25/03/16
Modify the API Gateway to collect metrics such as a the start and end time of each service call and write tehn to a DB		
Create a service which retrieves that data		
TEST-6: E2E System Test	28/03/16	30/03/16
A series of E2E tests written using Selenium to test the flow of the application from start to finish		
Sprint 6	04/04/16	22/04/16
TEST-7: Acceptance Test	04/04/16	04/04/16
An acceptance test with the project sponsor Emma O'brien of Mcor Technologies		
DOC-1: Final Project Documentation	05/04/16	22/04/16

Project Plan Gantt Chart



Appendix C: Requirements Elicitation Transcription

The text below is a copy of the correspondence from Emma O'Brien, reiterating the desired requirements elicited from our meeting. Although this system is not designed to specifically address one use case, the requirements gathered from this session were used to guide the design of the overall system.

Hi David,

Following on from our recent meeting, these are the features that I think would be really beneficial to an app that someone could use in my line of work.

The app would allow me to enter in a place name and generate business data for that location. So, for example, if I enter "Skerries, Co. Dublin", the app would give me a list of all the businesses in that area. Also, it would be great if I could search by town, city or province.

The details I would be most interested in generating would be business names, addresses, phone numbers and the latitude and longitude for each address. This is important to us as it would be a great help to our field sales representatives out on the road. Would it be possible to include Eircode information? If so, that would be brilliant.

In regards to the data that's generated, we upload leads onto our internal system via spreadsheets so it would be brilliant if a spreadsheet with all the data could be downloaded at the end.

Finally, it would be very important that the app is easy and intuitive to use as technical training would not be an option for many people in my position.

I wish you the best of luck in your project and please let me know if I can offer you any help. Looking forward to seeing the end result!

Kind Regards,
Emma O'Brien

Appendix D: Monthly Journals

Reflective Journal

Student name:	David Brady
Student number:	x12112267
Programme:	BSc. in Computing
Specialisation:	Cloud Computing
Month:	September

Starting back at college after being out in the workplace for the past nine months was certainly a shock to the system. Despite having time to come up with ideas for the final year project, there is always the sense that you could have done more.

I had decided on a structure regarding how I would manage this project a number of weeks prior to my return. Built into my plan was "sprint zero" which would be three weeks of research-related tasks aimed at gathering a better understanding of the technology stack I was planning to use.

Due to a number of comprehensive and time-consuming report assignments issued to us at the early stages of the academic year, a lot of time I had allocated for project research was used to complete these tasks.

During this time, however, I was able to cement my proposed idea down and identify the technology stack needed to deliver it. I feel confident that I will be able to start physically programming on Monday 5th of October which is the official start of sprint one.

My project proposal has now been submitted and I am looking forward to receiving some feedback.

I have not yet identified an appropriate supervisor for this project as I am unaware of any faculty with experience in my chosen technology stack. However, I feel that I would most benefit from a supervisor with notable project management skills, so I may look into faculty members delivering modules related to project management.

In terms of technical consultation, a number of senior developers from my work placement host company Workday, Inc. have offered any support they can in terms of advice and direction (obviously without contributing any code).

I look forward to next month when hopefully the first stage of my MVP will be built, tested and deployed.

Reflective Journal

Student name: David Brady

Student number: x12112267

Programme: BSc. in Computing

Specialisation: Cloud Computing

Month: October

October was a month of blocker after blocker which caused several delays. The first of these delays was in relation to hardware. I had been working off my personal laptop after returning my work machine to Workday, Inc. in September. My personal machine was greatly lacking in terms of processing power and memory which made using my proposed technology stack very slow and difficult. I attempted to solve some of these issues by using a cloud-based IDE for my development, however this brought about its own problems as my chosen development framework was not directly supported. I attempted to use the NCI machines which also proved to be too underpowered for my task. On top of that, my account for the NCI machines was locked out for an undetermined reason and so I could not use these machines for over a week.

In the end, my only solution was to purchase a new machine with the hardware specifications I required (as upgrading my previous machine would still not solve the hardware issue).

Luckily, the new machine has eliminated my previous hardware issues but I have lost out on two weeks' development time.

Another blocker I have experienced was the large number of assignments from other modules running in parallel to my final year project. Although these assignments are aimed at supporting the development of my final year project, the frequency of them and the time which they require is blocking me from achieving the development velocity I had hoped for.

I have made some progress in terms of the project's Geo Tagging Service as it is an integral part of my project. The reason I have started building this service as opposed to any other is that it is a viable standalone product. This is a type of insurance for me. If, in a few months' time, I discover that perhaps the scope of my project was too large, given the time allocated and the technology I need to learn in order to deliver it, I want to be able to roll back to something viable. I have been building this service as a standalone application using the Play Framework and Scala. Given the loss of time due to the hardware issues mentioned above, I have had to greatly utilise existing libraries/APIs relating to the Google Docs interaction. This, I believe, is justified as there is little point reinventing the wheel unless I have to. My focus at the moment is to have this service finished, deployed and tested as soon as possible so that I can move onto the remaining services.

Through my own study as well as material covered by both the cloud computing and network programming and distributed systems modules, I have been learning more about SOA and microservices. I feel that developing an application that utilises a microservices architecture rather than just a novel application will be far more beneficial to my career upon graduating from NCI. Therefore, after I have completed the Geo Tagging tool, I am considering making a shift from the Play Framework to Spray.io as it is a lightweight toolkit geared towards the development of microservices. As my Geo Tagging services are written in Scala, it will be easy to transfer them to the new approach if I choose to do so.

Finally, this month I have been appointed my academic supervisor of choice, Manuel Tova Izquierdo. Manuel is my cloud computing lecturer and has an extremely impressive level of experience behind him. I feel that I will learn a lot from his guidance and he will take great interest in my work. I have arranged to meet him shortly to demonstrate a prototype of my Geo Tagging Service.

Reflective Journal

Student name:	David Brady
Student number:	x12112267
Programme:	BSc. in Computing
Specialisation:	Cloud Computing
Month:	November

November was a very difficult month for me due to an unfortunate incident which resulted in me being very badly injured. I was medically unfit to work/study for the second half of the month. The college has been made aware of this issue.

I did, however, get work done that will benefit me in terms of my final year project. As part of my network programming and distributed systems module, I was required to take part in a group project in which we were required to build a distributed system with three distinct services.

I assigned myself the role of developing a data visualisation service built using AngularJS. This has given me some great experience as I am now able to build a completely separate client application which can interact with my proposed spray.io application via HTTP requests.

The backend of my application, which will be built in Scala with a spray integration layer, will make its components available via Restful Web Services to an AngularJS client application.

I also made progress this month on the Geo Tagging Service which is near completion. Unfortunately, I ended up building it as a standalone Ruby on Rails application as I was familiar with some Ruby Gems that could assist me in my development, especially in terms of the Google Sheets API. I will try and convert this to Scala as soon as I can.

Reflective Journal

Student name: David Brady

Student number: x12112267

Programme: BSc. in Computing

Specialisation: Cloud Computing

Month: December

During the month of December, I made a decision to change my proposed final year project significantly. I decided that, instead of a Geo Location-based "Deals" web application, I would now develop a system composed of independent microservices which could be used together in order to dynamically generate marketing data for an end user.

I decided to make this change for a number of reasons. Firstly, I had explained the Geo Tagging component to a member of the Marketing team at Mcor, who stated that such a service would be beneficial to them for the purpose of field sales lead generation. Secondly, I felt that my newly proposed project would suit my personal goal of using Play, Spray, Scala, Akka and Docker under a microservices architecture pattern better.

I set up a meeting with Emma O'Brien of Mcor to discuss my proposed project and received positive feedback. I elicited a number of requirements from Emma which has helped me to shape the design of the new project.

A downside to this change, however, is that it has added to the level of research required for the project which has taken up a significant amount of time. As well as this, I have had to make modifications to my existing supporting documents such as requirements specifications, proposals, etc.

This month, I also completed the Geo Tagging Service. The purpose of this service is to read address data from a remote spreadsheet on Google Drive, look up the longitude and latitude for each constituent address and then write them to the spreadsheet. I successfully developed and deployed a prototype of this application using the Ruby on Rails framework, however I wish to adapt it further to fit in with the overall project.

Reflective Journal

Student name: David Brady

Student number: x12112267

Programme: BSc. in Computing

Specialisation: Cloud Computing

Month: January

Like for many of my peers, the first half of January was focused on preparing for the Christmas exams. The second half of the month, however, gave me time to work on my project and prepare a working prototype of the system in time for the mid-point presentation scheduled for 10th February.

I developed a working prototype of the system's "Base Business Details Service" using Scala, Akka and Spray. This service provides a variety of endpoints that can be accessed via HTTP requests and return a set (or subset) of business records. The records are stored in-memory to improve speed and to allow for efficient Unit Testing. These resources are then returned to the user as JSON, using the RESTful design principle.

In terms of code coverage, I have created a series of unit tests using the "ScalaTest" framework and the FlatSpec trait. These tests only assert the expected behaviour of each function. I will implement functional end-to-end tests using Selenium at a later date.

The raw JSON response can be viewed in the browser by simply entering the URL endpoint, however I developed a user interface using AngularJS to allow the user to access the service resources and visualise them in a beneficial way.

I ran into a number of difficulties during the development of this prototype, the primary one being knowledge constraints surrounding the development language of choice. Fortunately, this knowledge barrier will lessen over time.

Another difficulty I experienced was when I tried to retrieve the resource from the server and display it in the UI. The cause of this problem was restrictions concerning Cross Origin Resource Sharing (CORS) as both the UI and Web Service were running on different ports. Fortunately, there was an available Scala directive which I could import into my project to handle CORS support. It was also fortunate that I discovered this issue early on as CORS support will be essential once each service is deployed.

This month, I also prepared a large part of my technical report documentation for this project. The report currently consists of 45 pages and covers topics such as requirements and system architecture. This document will be submitted prior to my mid-point presentation and will also serve as a base for my final report.

Reflective Journal

Student name: David Brady

Student number: x12112267

Programme: BSc. in Computing

Specialisation: Cloud Computing

Month: February

During February, I had my mid-point presentation. This gave me an excellent opportunity to demonstrate both the idea behind my project and my progress to date. The presentation went brilliantly and the lecturers I was presenting to seemed very impressed by my efforts. I was awarded an outstanding grade of 99% for the presentation which has really put me in a positive position moving forward.

For the rest of the month, I was focused primarily on developing the remainder of the composite services as well as the client user interface to interact with each service.
Having completed one service already, I had a better understanding on how to implement the other services in terms of HTTP integration layers and data serialisation, etc.

I had some issues in terms of JSON parsing for the Geo Tagging Service. As I am not using a framework such as Play, I had to use a third party library: "lift-web". Although I achieved what I wanted in the end, I found that the parsing options were unnecessarily difficult and the documentation was limited.

This month, I also got started on the process controller service. This is proving to be the most challenging service to date in terms of complexity. However, it has also been the most rewarding service to design and develop.
I have taken a very functional approach to the main algorithm, whereby I have decomposed as many of the steps as I can into individual functions. This has made the code more reusable, manageable and readable. The benefit of this is that the speed and efficiency of the system should be increased despite the extra function calls. Also, it will be easier to identify a point of error should one occur.

Reflective Journal

Student name: David Brady

Student number: x12112267

Programme: BSc. in Computing

Specialisation: Cloud Computing

Month: March

March proved to be the most productive month for me in terms of development. I had originally set a deadline to have all development and testing work done by the 1st of April. This was so I could focus on studying for my exams, deploy my project and write up my technical documentation ahead of the 29th April deadline (BETA upload date).

I have been successful in my goal and am satisfied that I have a viable product to present in May. I have been keeping in regular contact with my academic supervisor and he is also satisfied with the current state of the project.

I found that providing test coverage for this project was quite difficult for a number of reasons, the main one being that software testing, despite being a requirement on the final year project grading rubric, has not been covered during the previous four years of my degree course.
As Scala is my primary development language for this project, I adapted the ScalaTest framework for my Unit Tests (White Box). These ensure that each of the primary functions within the individual services is producing the results they should be.
For my End-to-End (Black Box) Tests, I employed an AngularJS-based testing framework called Protractor. This allowed me to write automated browser tests to replicate human interaction and match the end results with expected values.

I ran into difficulty when developing the proposed Rails service for writing the values from the composite data object to a Google Spreadsheet. The issue was that Google has changed the Ruby Gem that I was using to handle the server-to-server authentication. This change was causing me to lose a lot of development time which I simply could not afford. I had two choices: either use the standard Client Authentication model (which would have unwanted changes in terms of user flow) or scrap the service. I eventually decided to scrap this service and replace it with a new one.

This new service is one that I feel better suits the system as a whole. The service takes the composite data object and writes it as a JSON file to an Amazon Web Services S3 bucket. The user is then issued a hyperlink which they can use to directly download the file to their local machine.
This is a much better feature as it permits the user to have greater flexibility with the outputted data.

This change did not come without some stress and heartache, unfortunately. After setting up the service, my AWS credentials somehow became compromised and my AWS account was hacked. The hackers managed to run up a massive bill of €7,500 on my account in less than 24 hours. Luckily, after two weeks of correspondence with Amazon, this issue has now been rectified.

I go into April confident in my project and now aim to focus on deployment and documentation.

Appendix E: User Manual

The following user manual is designed to guide a user through the running and testing of the MicroDG system on their local machine.

CLONING THE SOURCE CODE

In order to run the system on your local machine, you must first acquire the source code. To do this, simply clone the source code from the MicroDG GitHub repository.

a. Ensure that you have Git installed on your local machine.
b. Open a terminal window.
c. Navigate to the directory where you want the source code to reside.
d. Use the following six commands:
    i. Git clone https://github.com/microdg/BETA-Base-Business-Details-Service.git
    ii. Git clone https://github.com/microdg/BETA-Geo-Tagging-Service.git
    iii. Git clone https://github.com/microdg/BETA-AWS-Storage-Service.git
    iv. Git clone https://github.com/microdg/BETA-Base-Business-Details-Service.git
    v. Git clone https://github.com/microdg/BETA-Process-Controller-Service.git
    vi. Git clone https://github.com/microdg/BETA-Client-UI.git

Once each repository has been cloned to your local machine, open five terminal windows. Now, navigate into each service directory.

In the next step, you will carry out a clean compile on each service.

COMPILING THE SERVICES

While in the root directory for each service (except BETA-Client-UI), enter the following command to run a clean compile:

a. sbt clean compile

Now that each service has been compiled, you can either run or test the services.

TESTING THE SERVICES (UNIT TESTS)

From the root directory of each service, enter the following command to execute the unit tests:

a. sbt test

RUNNING THE SERVICES

1. From the root directory of each service, enter the following command to run the Scala services:
    a. sbt run
2. Alternatively, you can open each service in the TypeSafe Activator UI and run the tests from there:
    a. activator ui

STARTING THE CLIENT UI

The user interface for this system is primarily a collection of Bootstrap styled web pages which use the AngularJS library to add additional functionality. Therefore, you can simply open the user interface in any web browser (Chrome is recommended).

However, if you wish to run the end-to-end protractor tests, it is essential that you know the IP address that the browser will use to serve up the web pages. My recommendation for this is to download the "200 Ok!" browser plugin from Chrome. This plugin will allow you to serve up web pages on localhost port 8888.

TESTING THE UI (END-TO-END TESTS)

Once the user interface is being served up on localhost port 8888, you must open two additional terminal windows and enter the following commands:

1. Terminal one: Start the Web Driver server
    a. webdriver-manager start
2. Terminal two: Navigate to the BETA-CLIENT-UI/test directory
    a. Run the protractor tests:
        i. protractor conf.js

This will initiate a series of automated browser tests.

INTERACTING WITH THE SERVICES

Once all the services are running locally and the user interface is being served up on the correct port, you can navigate to the index page and take a number of actions from there.

The format for interacting with each service is similar with all services:

1. The user accesses the service via the service gateway.
2. The user is required to enter in some data, either an option from a dropdown menu or, in the case of the Geo Tagging Service, an address string submitted via an input field.
3. The user is then required to click a submit button. Once the service process has executed, the user will be presented with a table of data.
    a. The user will be updated with the progress/status of the service request via an alert box positioned above the input area.
4. The user can view runtime statistics from the system's home page by either refreshing the page or clicking on the button to retrieve sample statistics.

Appendix F: UAT Feedback Forms

UAT Feedback Form

Name: Steve Hall **Date:** 27/04/2016

Occupation: Software Test Manager **IT Literacy:** Advanced

ID	Scenario	PASS/FAIL
1	Navigate easily from the home page to each composite service dashboard	Pass
2	Retrieve all business listings from the BBDS	Pass
3	Retrieve business listings from the BBDS based on TOWN name	Pass
4	Retrieve business listings from the BBDS based on COUNTY name	Pass
5	Retrieve business listings from the BBDS based on REGION name	Pass
6	View a detailed view of a business record given an ID	Pass
7	Retrieve the latitude and longitude coordinates for an address via the GTS	Pass
8	Generate a composite business object of all businesses using the PCS	Pass
9	Generate a composite business object of all businesses, based on TOWN name, using the PCS	Pass
10	Generate a composite business object of all businesses, based on COUNTY name, using the PCS	Pass
11	Generate a composite business object of all businesses, based on REGION name, using the PCS	Pass
12	Download a composite business object from the cloud to the local machine	Pass
13	View runtime statistics on an analytics dashboard	Pass
14	Generate and view sample runtime statistics on an analytics dashboard	Pass

User Experience
Please rate between 1 – 5 (1 being low and 5 being high)

Ease of use	___5____	**Response time**	___5____
Speed	___5____	**Look and feel**	___5____
Usefulness	___5____	**Reusability**	___5____

General Feedback

Excellent user flow and response time

Future Improvements

None

UAT Feedback Form

Name: Frances Walsh **Date:** 04/05/2016

Occupation: Optical Assistant **IT Literacy:** Intermediate

ID	Scenario	PASS/FAIL
1	Navigate easily from the home page to each composite service dashboard	Pass
2	Retrieve all business listings from the BBDS	Pass
3	Retrieve business listings from the BBDS based on TOWN name	Pass
4	Retrieve business listings from the BBDS based on COUNTY name	Pass
5	Retrieve business listings from the BBDS based on REGION name	Pass
6	View a detailed view of a business record given an ID	Pass
7	Retrieve the latitude and longitude coordinates for an address via the GTS	Pass
8	Generate a composite business object of all businesses using the PCS	Pass
9	Generate a composite business object of all businesses, based on TOWN name, using the PCS	Pass
10	Generate a composite business object of all businesses, based on COUNTY name, using the PCS	Pass
11	Generate a composite business object of all businesses, based on REGION name, using the PCS	Pass
12	Download a composite business object from the cloud to the local machine	Pass
13	View runtime statistics on an analytics dashboard	Pass
14	Generate and view sample runtime statistics on an analytics dashboard	Pass

User Experience
Please rate between 1 – 5 (1 being low and 5 being high)

Ease of use	___5___	**Response time**	_5___
Speed	___5___	**Look and feel**	_5___
Usefulness	___5_____	**Reusability**	_5_____

General Feedback

Simple and intuitive navigation between services.
Very quick response times.
I found it very easy to use.

Future Improvements

None

UAT Feedback Form

Name: Laura Fay **Date:** 04/05/2016

Occupation: Student **IT Literacy:** Intermediate

ID	Scenario	PASS/FAIL
1	Navigate easily from the home page to each composite service dashboard	Pass
2	Retrieve all business listings from the BBDS	Pass
3	Retrieve business listings from the BBDS based on TOWN name	Pass
4	Retrieve business listings from the BBDS based on COUNTY name	Pass
5	Retrieve business listings from the BBDS based on REGION name	Pass
6	View a detailed view of a business record given an ID	Pass
7	Retrieve the latitude and longitude coordinates for an address via the GTS	Pass
8	Generate a composite business object of all businesses using the PCS	Pass
9	Generate a composite business object of all businesses, based on TOWN name, using the PCS	Pass
10	Generate a composite business object of all businesses, based on COUNTY name, using the PCS	Pass
11	Generate a composite business object of all businesses, based on REGION name, using the PCS	Pass
12	Download a composite business object from the cloud to the local machine	Pass
13	View runtime statistics on an analytics dashboard	Pass
14	Generate and view sample runtime statistics on an analytics dashboard	Pass

User Experience
Please rate between 1 – 5 (1 being low and 5 being high)

Ease of use	___5___	**Response time**	____5_____
Speed	___5___	**Look and feel**	____5_____
Usefulness	___3___	**Reusability**	____5_____

General Feedback

This system would be very helpful to someone who requires its function. The layout was very good and it was super easy to use.

Future Improvements

None

UAT Feedback Form

Name: Clare Reilly **Date:** 27/04/2016

Occupation: Primary School Teacher **IT Literacy:** Intermediate

ID	Scenario	PASS/FAIL
1	Navigate easily from the home page to each composite service dashboard	Pass
2	Retrieve all business listings from the BBDS	Pass
3	Retrieve business listings from the BBDS based on TOWN name	Pass
4	Retrieve business listings from the BBDS based on COUNTY name	Pass
5	Retrieve business listings from the BBDS based on REGION name	Pass
6	View a detailed view of a business record given an ID	Pass
7	Retrieve the latitude and longitude coordinates for an address via the GTS	Pass
8	Generate a composite business object of all businesses using the PCS	Pass
9	Generate a composite business object of all businesses, based on TOWN name, using the PCS	Pass
10	Generate a composite business object of all businesses, based on COUNTY name, using the PCS	Pass
11	Generate a composite business object of all businesses, based on REGION name, using the PCS	Pass
12	Download a composite business object from the cloud to the local machine	Pass
13	View runtime statistics on an analytics dashboard	Pass
14	Generate and view sample runtime statistics on an analytics dashboard	Pass

User Experience
Please rate between 1 – 5 (1 being low and 5 being high)

Ease of use	___5____	**Response time**	___5____
Speed	___5____	**Look and feel**	___5____
Usefulness	___5____	**Reusability**	___5____

General Feedback

None

Future Improvements

Include hyperlinks for services behind their images or titles on the home page.

UAT Feedback Form

Name: Paul Riordon **Date:** 08/05/2016

Occupation: Sales Manager **IT Literacy:** Intermediate

ID	Scenario	PASS/FAIL
1	Navigate easily from the home page to each composite service dashboard	Pass
2	Retrieve all business listings from the BBDS	Pass
3	Retrieve business listings from the BBDS based on TOWN name	Pass
4	Retrieve business listings from the BBDS based on COUNTY name	Pass
5	Retrieve business listings from the BBDS based on REGION name	Pass
6	View a detailed view of a business record given an ID	Pass
7	Retrieve the latitude and longitude coordinates for an address via the GTS	Pass
8	Generate a composite business object of all businesses using the PCS	Pass
9	Generate a composite business object of all businesses, based on TOWN name, using the PCS	Pass
10	Generate a composite business object of all businesses, based on COUNTY name, using the PCS	Pass
11	Generate a composite business object of all businesses, based on REGION name, using the PCS	Pass
12	Download a composite business object from the cloud to the local machine	Pass
13	View runtime statistics on an analytics dashboard	Pass
14	Generate and view sample runtime statistics on an analytics dashboard	Pass

User Experience
Please rate between 1 – 5 (1 being low and 5 being high)

Ease of use	___5___	**Response time**	____5_____
Speed	___5___	**Look and feel**	____5_____
Usefulness	___4___	**Reusability**	____5_____

General Feedback

I really like the idea of this application and think that it could be hugely beneficial to sales teams once the functionality is extended.

Future Improvements

More services