Jack Davey

**Project Plan for Testing and Tailoring Cloud Storages**

**Milestones**

There are three major milestones that I hope to achieve in the duration of my MSC Project:

**Milestone 1**

This milestone concerns the implementation of eventual consistency over a standard ACID Database. This will be implemented in iterations 1 through 4.

**Milestones 2**

Milestone 2 will be devoted to adding the second BASE property of eventual consistency to the system. This will take up iterations 5 through 8.

**Milestones 3**

Milestone 3 will be devoted to running the sample programs that I have developed and running them on a real cloud system.

**Iteration 1 – Getting used to the System – complete (12th June 2015)**

**Start Date: 8th June 2015**

**End Date: 14th June 2015**

**Objective 1**

**Allow the user to perform basic creation and deletion of data in the database.**

**Objective 2**

**Allow the user to insert, update and delete data from existing tables**

This should work by the end user being able to pass in an SQL query as JSON and have this executed by the database. The results should then be sent back as another piece of JSON. This feature will not implement eventual consistency at this point, and is partly here to allow me to ensure that I am comfortable with the tools that I am using to complete the project.

**User Story 1**

As a user of the system, I should be able to pass an SQL query encoded in JSON as a web service request. This should be able to be read by my application.

**User Story 2.**

As a user I should be able to get a feel for the form that the rest of the application will take just by looking at this user story.

**User Story 3**

As a user the effects of running the query on the database should be returned to me when I run the application through as JSON. In other words if I pass in the query “select \* from cars” then the entirety of the cars table should be returned.

**User Story 4**

As a user, If I send the system an invalid request, then it should be rejected and returned to me. As an example, if I send this service, “select 33 from cars” then an error message should be returned in JSON format.

**Iteration 2 – Implementing Eventual Consistency – complete( 24th June 2015)**

**Start Date: 15th June 2015**

**End Date: 21st June 2015**

**Objective 1**

**Allow the user to insert, update and delete data from existing tables. Eventual consistency must be active across all of these operations.**

This objective will involve the user passing in an SQL query as done in JSON before. I will most likely modify the format of the JSON to make it easier to parse. Rather than being sent straight to the database as before, the query will be stored in the application for a period of time. Once this time has expired, the application will then store the query in the database, following the procedure for achieving eventual consistency as outlined in my design.

**User Story 1**

A user should be able to encode the data in a specific kind of JSON and then have the application convert that JSON into an object that can be run on the database. In other words, if the user entered a valid sql update statement, then this should be converted into JSON in the application .

**User Story 2**

An application should be able to persist these objects in the database for a specified period of time.

**User Story 3.**

The data should be spread across multiple servers within the application. In other words, if the user sets the age of the person ‘dad to 42 in the database, then it should either return 42, or the original age depending on what was accessed.

**User Story 4**

At some point, (normally 100 seconds) all these updates should be sent to the database, and any consistency issues should be resolved by running the consistency algorithm detailed in the design document. Thes updates should then be fully consistent, ( any queries on the database should return the new value.

**Objective 2**

|  |
| --- |
| **Allow the user to synchronize the system (make everything fully consistent)** |

This should be fairly straightforward to implement, as all it involves is providing the user with a mechanism by which to make the system fully consistent without waiting for the system to do that by itself.

**User Story 1**

A user should be able to send a request to a specific web service.

**User Story 2**

Once a request is received the consistency protocol should be activated, and there should be no stale data still inside the application.

**Iteration 3 – Customization of eventual consistency and preparing for the early deliverable. – complete (26th June 2015)**

**Start Date: 22nd June 2015**

**End Date: 28th June 2015**

**Objective 1**

**Write a web service that allows the user to change the amount of time the system waits before making everything consistent**

This web service should take a single integer parameter. It should then change the parameter as described above. The next scheduled event should still proceed as planned, however. Events should only be scheduled one at a time. As an example, if the system is due to be made consistent every 100 seconds, and 200 is passed into this service, then all future consistency runs should happen every 200 seconds.

**User Story 1**

There should be a web service designed to achieve this functionality that takes a single integer parameter as part of the URL.

**User Story 2**

When a request is sent to this service, the time between consistency runs should be changed.

**User Story 3**

Invalid parameters, such as negative numbers, should be rejected.

**User Story 4**

The user should be informed of whether this was successful through a JSON result.

**Objective 2**

**Catch up and ensure everything is ready for the early deliverable.**

I am allowing significant time this week to ensure that everything is ready for the early deliverable which is due straight after this iteration ends.

**Iteration 4 - Getting information on the system – complete 3rd July**

**Start Date: 29th June 2015**

**End Date: 5th July 2015**

**Objective 1**

**Add a web service that shows the current consistency status of the system and also add customization options.**

This Objective will involve writing a piece of functionality that allows the user to look at the pieces of data that are currently within the system. For each piece of data, the web service should show what the change is (such as updates, deletes, inserts) etc., as well as the time at which it is due to become consistent with the rest of the data.

**User Story 1**

A user should be provided with an appropriate web service to use to carry out the request.

**User story 2**

When this web service is called all the inconsistent updates currently in the system should be returned in JSON format.

**User Story 3**

The time when all of these requests are due to be made consistent should also be displayed as part of the JSON request above.

**Objective 2**

**Perform an evaluation of the prototype implementation of eventual consistency and improve upon it.**

At the beginning of the week, I will have submitted my early deliverable for the project. I would therefore like to use a significant portion of this week to evaluate the prototype for this week and improve upon it. Part of this evaluation will be done using the sample programs I am developing this week.

**Objective 3**

**Research common algorithms for basic availability.**

I will also spend some time this week going through all the different approaches for achieving basic availability that I have identified in my research and choosing the model base that I would like to implement. I would aim to modify my design document showing how I would do this.

**Objective 4**

**Write some sample programs that show that the system is working.**

This story will involve writing some small test applications to ensure the application is working as it should be. I plan to use code from some of the other functionalities to achieve this goal.

**User Story 1**

The user should be provided with a program to count the number of attempts it takes to receive a number of inconsistent results.

**User Story 2**

Another program should be created that does the same job with two writers.

**User Story 3**

A program should be written to count the number of inconsistent results in a set period of time.

**Iteration 5 – laying the groundwork for basic availability Completed (19th July 2015)**

**Start Date: 6th July 2015**

**End Data: 19th July 2015**

**This iteration is longer than the others, due to the fact that I will be on holiday from the 7th to the 14th July.**

**Objective 1**

**Provide a mechanism for the user to change various parameters about the system.**

This objective is all about moving the facilities for changing parameters of the system into their own object. It will also provide a more generalized web service to control this functionality.

**User story 1**

A web service should be provided that allows the user to pass in the name and value of the parameter they wish to change.

**User Story 2**

All other occurrences of these parameters should be provided by the backing object for this service.

**User story 3**

If a user or the program tries to access a value that does not exist, an exception should be thrown, and this should be reported to the user.

**User Story 4**

The service should reject negative numbers or numbers that are above a certain size.

**User Story 5**

When a valid key is entered, then the value should be returned.

**Objective 2**

**Implement a mechanism by which a replication server can fail to process a request in order to simulate the server not being available.**

**User Story 1**

When an update reaches the replication servers, a decision should be made as to whether that message can be serviced or not.

**User Story 2**

Allfailed updates should be resent back into the system in the normal way, and should eventually succeed.

**User Story 3**

If an update fails, then the user should be able to find this out., through the calling of a separate web service.

**User Story 4**

Once an update fails, then the server should be down for a certain period of time that can be user customizable. All updates that tha server was currently carrying should be classed as failed.

**Objective 3**

**Split the existing replication server into a replication cluster with a master node and several slave nodes.**

This objective is all about implementing the infrastructure for the basic availability section of the application. Each replication server will do the same jobs as it did before, but only the master will be allowed to do key things such as sending all the records to the database for committing purposes. The slaves just keep a record of all data sent, in case one of them becomes the master.

**User Story1**

Make the change by creating a new replication cluster actor that controls all the replication servers.

**User Story 2**

The components that used to talk to the replication servers, the overseer and the marshaller, should see no difference.

**User Story 3.**

One of the replication servers should be a master, that should handle all the goals previously done by the original replication server.

**User Story 4**

Only the master node shoud be allowed to send messages to the toher processes in the network.

**Iteration 6 – Implementing basic availability behaviour**

**Start Date: 20th July 2015**

**End Date: 26th July 2015**

**Objective 1**

**Implement the voting strategies discussed in the design document so that one of the slaves can become the new master if it goes down.**

This objective is concerned with implementing the interactions between the various master and slave processes so that they actually implement the replication strategy, discussed in the design documents.

**User Story 1**

Slaves must continually contact the master to see if it is still alive, if not then an election must be held to elect a new master.

**User Story 2**

When the master goes down, all of the slaves must decide amongst themselves who the new master is.

**User Story 3**

Once a master is chosen, then the old master must become a slave.

**User Story 4**

These changes should have no impact on the rest of the system.

**Objective 3**

**Allow the user to customize the amount of basic availability provided to the end user.**

This could be through changing the amount of data a server could have at any one time, or changing the frequency at which errors occur.

**Iteration 7 - Evaluating basic availability**

**Start Date: 27st July 2015**

**End Date: 2nd August 2015**

**Objective 1**

**Produce some sample applications to show the system working.**

As I haven’t decided how I’m going to implement basic availability yet, I cannot say for certain what I am gong to do here. It will have similar aims to the code implemented for eventual consistency, however.

**Objective 2**

**Do some research on cloud systems to see if I can figure out if I can run my sample applications on them to get the same effects.**

Because of the very nature of cloud systems, I want to do some research on cloud systems to see if I can run the examples I got on them for comparison like I did with eventual consistency. Due to the nature of basic availability however, this might not be possible.

**Objective 3**

**Modify the sample applications produced for eventual consistency so that they run on a cloud infrastructure rather than on my system.**

A large part of this story will involve working out what needs to be changed with my samples so that they can run on a cloud service. This might take some time, as a good bulk of the work will involve learning how to use the APIs provided by the cloud service effectively.

**User Story 1**

These applications should be additional web services accessed through my application.

**User Story 2**

They should produce exactly the same outputs as if they had been run from my own store.

**Iteration 8 - Evaluation**

**Start Date: 3rd August 2015**

**End Date: 9th August 2015**

**Objective 1**

**If possible modify my programs to demonstrate basic availability to run on a real cloud system.**

This may not be possible but I would like to do this to help evaluate the implementation of basic availability.

**Objective 2**

**Compare the sample applications to my system and try to gauge how effective the system is at simulating eventual consistency.**

I would do this by running the various programs and comparing how long they take to reach the same results. Another possibility is for me to look at the frequency by which the two systems return inconsistent results.

**Any remaining time left will be used to catch up with anything I haven’t done and will be used to make sure my project is as polished as can be.**