Balance API Creation on AWS

Project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering/Information Technology

by

Ayush Patel (161478)

Under the supervision of
Rizwan Ur Rehman

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

Candidate's Declaration

We hereby declare that the work presented in this report entitled **"Balance API Creation on AWS"** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Information Technology** submitted in the department of Computer Science and Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from March 2020 to May 2020 under the supervision of **(Rizwan Ur Rehman)** (Assistant Professor, Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Ayush Patel (161478) 

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Rizwan Ur Rehman. Gaurav Foujdar

Assistant Professor Mentor

Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology

Dated: 25/05/2020

ACKNOWLEDGEMENT

Foremost, we would like to express our sincere gratitude to our project guide **Rizwan Ur Rehman (Assistant Professor)** for the continuous support in our project, for their patience, motivation, enthusiasm, and immense knowledge. Their guidance has helped us in all the time of this study and writing of this report. We could not have imagined having a better advisor and mentor for our project. We would also like to thank them for lending us their precious time.

I would also like to thank my Mentor Gaurav K Foujdar for his help in making me understand the systems and providing support all the way throughout the Project.

We are also very thankful to all the faculty members of the department, for their constant encouragement during the project.

Last but not least we would like to thank our parents, who taught us the value of hard work by their own example.

Date: 25/05/2020…

Ayush Patel (161478)

Contents

List of Figures

List of Abbreviations

AWS – Amazon Web Services

REST – Representational State Transfer

API -Application Program Interface

SDLC- Software development Life Cycle

SIT-System Integrated Testing

SOA- Service Oriented Architecture

ABSTRACT

Legacy based Services are old and run on old technologies that are much difficult to maintain and if an issue occurs in a legacy service, the customer can face a longer downtime which is a bad customer experience. The connections to Database using these Services to the Database takes a longer time and again contributes to a bad Customer Experience in terms of Longer Time to Load a particular Page.

AWS Stack comes as a Solution to this. Not only is it easy to maintain and has more Availability and Reliability but connections to databases take considerably less time than using Legacy Based Architecture.

The pros of using this is a reduction in latency of Service, a Higher Availability and a lower Downtime for The Service and maintenance becomes a lot easier.

And for these reasons we will be our Service would be created on the AWS Stack

CHAPTER 1: INTRODUCTION

1.1 Introduction

As a customer the one thing that we look forward while using any service or any part a service is that it is fast enough even on a high latent network connection. Another thing that we look forward is that when we hit the service ,it should be available. No matter how fast a service is if it has a high failure ratio the customers are not likely to use it. While Designing these services we should also think about Scalability. Even a small change in Production is presented to potentially millions and millions of customers and a downtime as low as a minute is a bad idea. Furthermore there are bound to be issues in Service at some point which needs to be fixed. We should have the ability to quickly rollback the changes/mirror the service onto another Host while we are fixing the issue or doing some internal migration/update on the Service

Current Legacy Services take a lot of time to be replicated onto a different host that can serve the customer with traffic while we investigate an issue. The setup itself can take up hours. Most of this setup and could be easily avoided if we move the current service stack to AWS as it has most of the automation already in place to deal with these kind of issues. Setup also is more or less a one click kind of setup.

While doing the migration ,we should ensure that there are no internal or external teams that are currently using the current APIs in the legacy services face any issues. The newly designed APIs in the new service should be able to mirror the Request /Response parameters of our Original APIs in the legacy Service so that no changes are required on the partner team's side in their current implementation as this will cause a chain of changes that will affect a huge number of partner teams and all of them will not be aligned to go forward with the changes that are proposed by our team. Since the migration will take a considerable amount of

time, separate hosts needs to be setup which have the Replicated Deployment as in our current Production servers ,while the current hosts keep serving the customers and all partner teams till we are done with the migration

Request Response structure for Update Balance

Request:

{

Operation: UpdateBalance

TableName: Tables where Balance Information is stored

customerID: customerId of person trying to add the money

payment method: Payment method for the current transaction (Credit card/net Banking/UPI)

}

Response:

{

Transaction Status: Success/Failure

Balance: Updated Balance for the customer post transaction

OrderId: A Unique OrderID associated with the transaction

}

Request Response structure for GET Balance

Request:

{

Operation: GetBalance

customerID:customerId of user whose balance needs to be known

}

Response:

{

Transaction Status: Success/Failure

Balance: Updated Balance for the customer post transaction

}

Tools and Technologies Used

AWS is the Amazon's Cloud Services much like Azure or Google Cloud Platform. It Provides a large range of services like Compute, Storage ,Security and Scalability. The Services running on AWS can be easily Scaled Up or Scaled Out based on the traffic and compute requirements. It provides easy connection with efficient Databases Like DynamoDB

DynamoDB

DynamoDB is a NOSQL database that has a major advantage over most other NOSQL Database Services. One major advantage of using this is that we don't have to worry about restructuring our Service when we start to receive a higher Traffic than we anticipate. It can be automatically scaled up and frees us from worrying about hardware side of things. It is also very reliable due to replication of data onto its clusters. In case if a single node falls ,the service request is forwarded to another node holding the data. Another major concern while using a particular DB is from a security standpoint. DynamoDB Provides encryption at rest . It can handle a large amount of traffic (39000TPS) very easily. Any tables created can be restored back to any commit back to the last 35 days. It provides integration with all major languages like Python, Ruby, Java etc

Apart from the sheer Security, Reliability and availability, DynamoDB provides latency reduction mechanisms using Caching Mechanisms like DAX

DAX

DynamoDB Accelerator or DAX is an in memory Caching System that can provide read speeds in microseconds. In Scenarios like these the Latency of a Service mainly depends on the network latency because the read, write and update latency is almost negligible. Another major Benefit of using DAX is that standard API calls to DynamoDB Do not need to be Modified in any way. By adding DAX

to our DynamoDB instance AWS automatically ensures the managing and fetching data from DAX when a call to DynamoDB is made

Due to its very low latency features DAX is being used by customers like Tinder, Canon, Twilio and Expedia

Since our Service involve High Reads and high write operations ,We will try to use DAX in our DynamoDB Application.

Kinesis Stream

Kinesis Data Stream is a real Time Data Capturing and Streaming Service and can be used to take thousands of bytes of data from here to there within minutes .It can be used for Real Time Data Analytics and collecting data from millions of devices.

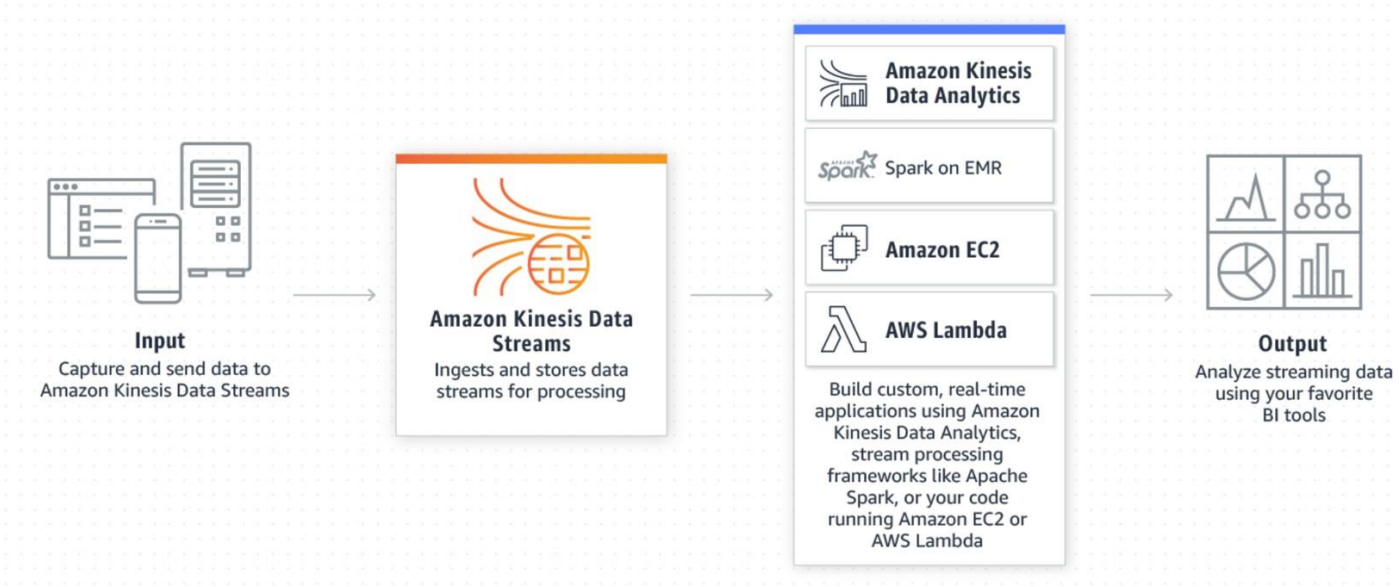

FIG 1:KINESIS STREAM

AWS Lambda

AWS Lamda is a serverless compute service that runs our code without us having to worry about anything else. There is an out of the box Support for Java, C# Python and Other Programming Languages. It deploys all our code, and we don't have to worry about adding more servers etc as the usage starts to increase. It takes care of maintenance. Most of the things are automated that can be monitored using CloudWatch. While this is an extremely useful resource it is a very costly one too ,So we should try to find optimizations wherever possible and try to reduce the number of lambda functions by as much as possible

Git

Git is a version control System that is used to check the status of files overtime and provides capabilities like Rolling back the changes to a previous version in case of any issues with the current changes and provides the capability to allow Multiple people working on it

Here is how it works as explained by a Sequence diagram

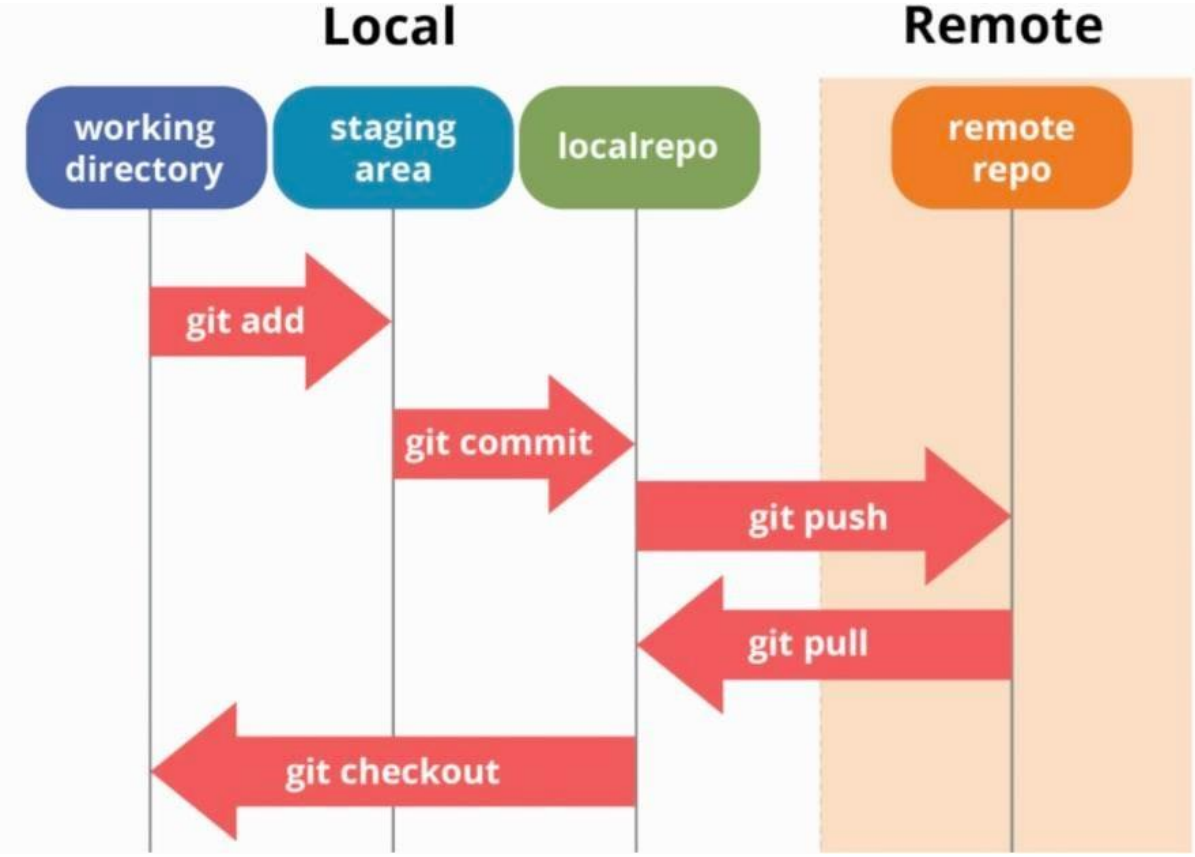

FIG 2: GIT OPERATIONS SEQUENCE DIAGRAM

The working Directory is where we are currently making the changes, i.e the location of the current file that we are working on. Before we can push our changes to the git remote ,we need to get those changes into a staging area which is essentially tracks what changes were done to the file and creates a difference between already existing files in the directory vs new things added or deleted post last commit.

Post this a git commit helps to commit the changes that we have just made , create a version difference and add it to the history of previous changes that we have done and get it ready for the deployment onto a centralized repository system like Github.

The main benefit that comes from using git is for a Very Large Single project that requires some common base dependencies is its ability to allow people to work on different branches

Since we do not want to pollute the mainline which pushes features already used by our customers. We can create a separate Branch and test all our features on to that branch and once we are sure that it won't break any existing feature and is working fine we can merge it finally into the master branch and Deploy it as a new Feature to the Customers

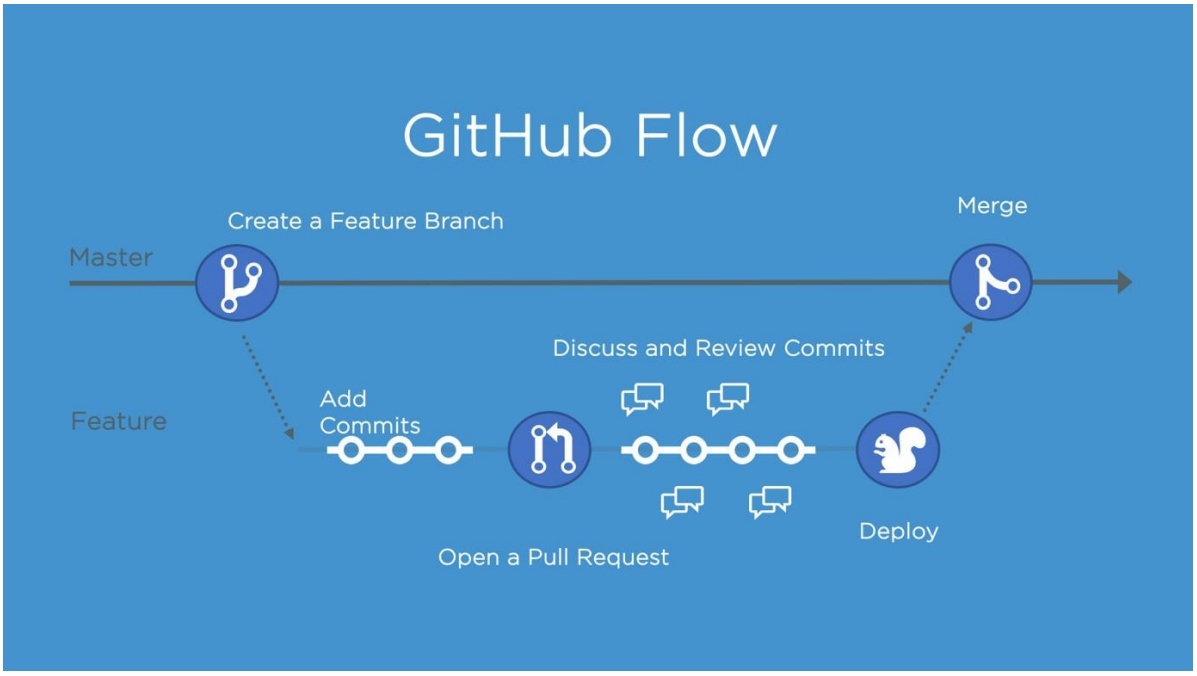

Fig 3:Git Branching

Language/Framework Used:

Spring Framework

Spring is a very popular Open Source Framework built on top of Java. It can be used to develop Applications based on Java language very easily and in a fast and efficient manner. The one major good thing of spring is its Modularity Nature.

Instead of Designing whole Big Services we can break it down into small small modules that can be Reused. Ideally We want as little Interdependency of modules as Possible.

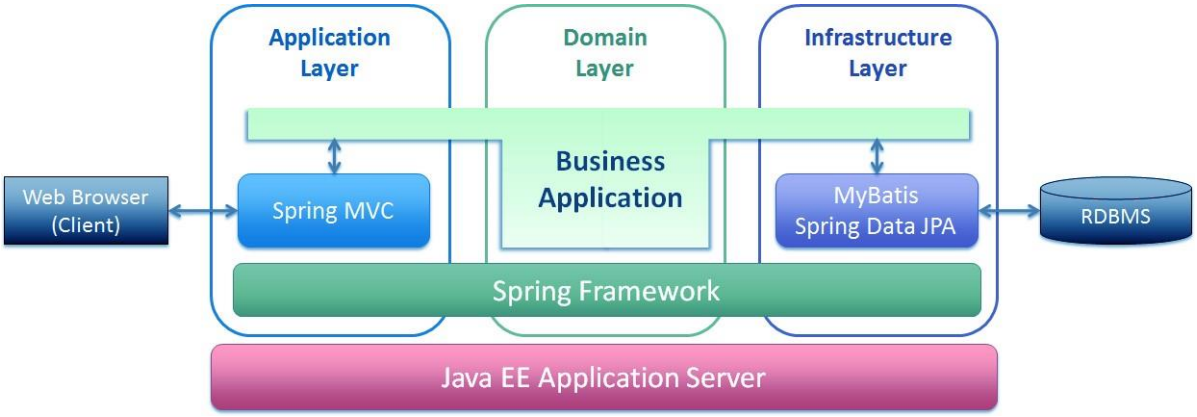

Fig 4: Spring Framework

Whatever Application or API we build ,we want to divide it into 3 Subsections

Client Layer:

The one That Actually makes a request to our API and is mainly Responsible for Displaying the data onto UI to the Client and makes request to the Backend Architecture. It contains absolutely No Data and No Business Logic. It is more of an abstract layer that can be used to just display and render the final state of the data post Business Logic

Business Logic Layer:

This is the Layer that is actually Responsible for the Processing of the Data. It takes in Data From the Database and according to our needs it Processes it and converts it into a form where it can be send to the Client Layer To Be Rendered.

Now the Logic of having this Business Layer Separately is that in future if our Requirements change, we can simply make changes to this Layer without

affecting any other Changes and thus No change would need to be propagated to any other Layers.

Database Layer:

This is the Layer which actually Stores the Data in a Secure Way .It is the authoritative source of data for the business layer to do its processing on and is usually maintained by an administrator. Having all these Three Layers, allows incremental changes to be delivered to customer without them having to do anything on their end. Ideally we want the customer to be able to use a feature as easily as possible and the separation of our entire Structure into Three Layers helps Achieve that. It is also good from a point of Security as the customers directly cannot access the data in any way or form other than calling our Backend Business Logic Layer which In Turn Calls the database layer

The main reason that we are going to use Spring is because of one particular feature that it provides called Dependency Injection

Why is there any need of Dependency Injection

Imagine a Scenerio where One Class needs some other class's object in some way or form to proceed with its Operation. This essentially means that class one is dependent on another class. Now while this dependency seems completely normal ,on a large scale System this Dependency can Be very Fatal. Imagine Having one component Break down in a real world Scenario could be the cause that entire System Goes Down. But at the Same Time its impossible to not use object of another class .Let us understand it a more by using the following Example

```
public class PublishLogstoDatabase(){

    Logger logger;
    public PublishLogstoDatabase(){
        logger = new Logger();
    }

    public void WriteToDB(){
        String final=logger.Log("About to do some work.").toString();
        //add the String to Database (Just an Example to help Understand)
    }
}
```

Fig 5: Without Dependency Injection Explanation

Why is this a bad Practice

1) So in this Example PublishLogsToDatabase class has a hidden dependency on Logger class. What that basically means is that this dependency is not that obvious without us having to peek deep into the code

2) If at a later stage we decide not to use this Logger Class but a completely Different one called Balance Logger. Now this would essentially mean changing that we will have to edit PublishLogstoDatabase class or other places where our previous Logger is Used. It could mean making changes to a large amount of files. Imagine if that class becomes deprecated for Some reason then whoever was using it would have to make changes in its class

3) To test out our PublishLogstoDatabase class ,we would need a functioning class and we cannot just do it if it hasn't been Implemented Yet.

Now lets Look at another example

```
public class PublishLogstoDatabase(){
    InfoLogger logger;
    public PublishLogstoDatabase(InfoLogger logger){
        this.logger = logger;
    }

    public void WriteToDB(){
        logger.Log("About to do some work.");
        String final=logger.Log("About to do some work.").toString();
        //add the String To Db,Just an example
    }
}
```

Fig. 6: Explanation Dependency Injection

In this Implementation instead of creating an object for the InfoLogger class, we need to pass it in form of a constructor argument. Now there are a lot of Observable Difference that can be seen in this particular version of Implementation

1) By just Looking at the Constructor of the PublishLogstoDatabase class, we can see that it has a Dependency on Other Class. Unlike in the last version where a deep dive is needed into the code to figure out this dependency, its pretty apparent in this one

2) InfoLogger is just an Abstract Interface, It can be easily swapped out with any other thing that has an InfoLogger type Implementation without having to change the code of PublishLogstoDatabase class

3) We can easily Mock InfoLogger and test our PublishLogstoDatabase class. That means we don't even need the other class to be fully functional to write the Tests for our current class.

It also allows us to swap different implementations of InfoLogger based on our Business Logic. So one way of Explaining the advantage of this in very plain and simple English would be to think about a remote and the battery that it operates on. Now the remote needs the battery to provide its functionality

and that is what could be termed as a dependency. Now Imagine a remote that only works on Brand X battery and if we want to use any other brand we would have to make changes to the structure of remote that. That itself is a problem. Compare it to what we use, we can swap battery of any brand in our remote and that works just fine. So this practical example could be used to understand a bit about why having a rigid coupiling is always a bad idea especially in a very large scale system

As we can see Dependency Injection covers some of the pit falls that we have when there is a not loose coupling between modules and they are dependent on each other via Rigid Class Implementations. Not only that it makes different people less working on different parts of a project less dependent on each other to deliver a fully functional and tested module that they are working on. We can easily mock out the result of the Dependencies and can even write cases verifying that they indeed are interacting with one another and have a much greater flexibility with our Modules. This also leads to a more cleanable and maintainable code and one indirect benefit of this could be the Single Ownership Principle wherein One person can entirely own a module and be responsible for the changes he makes in that module. If the entire API starts to behave Abnormally, like getting higher response time or Getting more Timeouts etc ,The module that is causing those issues can be easily tracked and the person who owns the module can work on fixing it without entire teams making changes to their respective modules.

An example flow of Injection in Spring Framework can be Seen Below

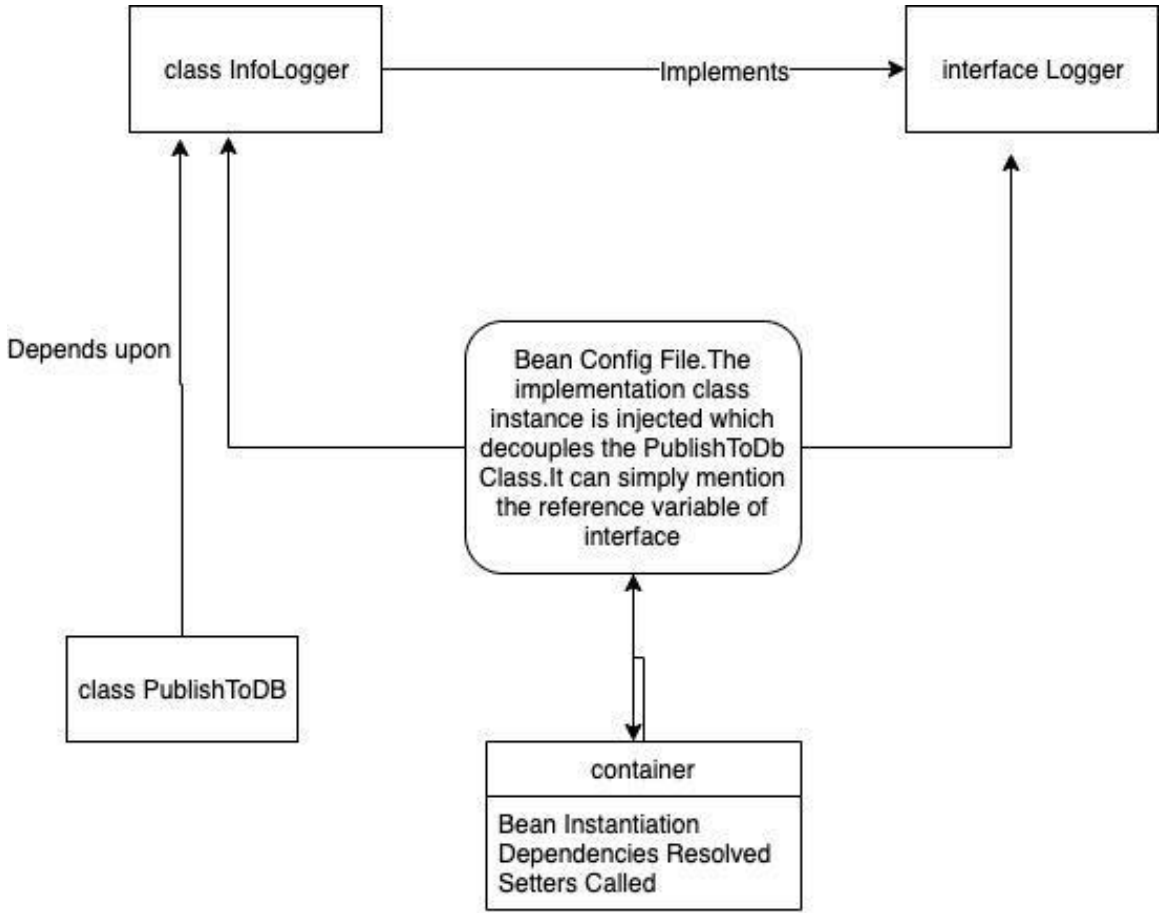

Fig7: Example flow of injection of class instance

1.2) PROBLEM STATEMENT

The Balance API and its operations are a very crucial part of any payment based Systems . The Basic Thing for any Payment System is the ability to be able to add and withdraw money using various payment modes. What is more important is keeping in constraints that people who will be using these APIs might also have a very slow and unreliable Internet and keeping all these things in mind ,our API should be able to handle a variety of error case Scenarios and should have a lower response time and a minimum of Downtime. Any Service or API that is created is bound to have some amount of Downtime, we should be able to minimise this. It should also be scalable for potentially millions and millions of Customers and provide accurate results.

Keeping these above constraints in mind, we should come up with an API that can seamlessly be integrated with other services while maintaining the above factors in mind. It should also have the capability to block requests from unauthorised services and should only expose the endpoints to services that have had registered to use this service. While designing we should also follow the best principles like having a proper variable name, breaking the modules into as smaller parts as possible and following a modular approach. The code that we create should be easy to understand and Debug and not only that its ownership should be easily transferable and should have a loose coupling with its dependencies.

1.3) OBJECTIVE

The main objective is to create a Balance API that can update and fetch the Balance of a Customer from the Backend while showing properties of low latency, high availability and an ability to handle an unusually amount of high traffic. It should be scaled enough to handle any major traffic spikes that come along and should not crash down in those situations.

It should have a very low downtime and should be easy to manage and provide an easy integration to services who register to use this API.The API should also block unauthorized services that try to access the Endpoints and shoudn’t allow any transactions without the authorization.This is to reduce any misuse that might be caused due to a third party service that is not supposed to access the API resources

While keeping the above things in mind,we should also come up with design that is viable in the longterm and makes any future API Integration seamless

1.4) METHODOLOGY

FIG 8: METHODOLOGY

We first start off by setting up the AWS Architecture that we will be Deploying our Service.

This mainly Involves

- Getting the console access to the hosts that we will be Deploying to
- Configuring the Code Deploy Console and setting it up
- Creating the instance
- Creating the Application onto the Instance
- Granting AWS Permissions to edit, and deploy the application
- Deployment and Iterative Development

We begin by getting ssh access to the hosts that we will contain our compiled files and other dependencies.This ssh access can be later used to log in to host and can be useful in a lot of scenerios like debugging,checking logs etc.Post that we need to specify the services that we need .for us the come in form of DynamoDb,Lambda and other similar services like stream.

Post having all these things setup we require to create our application onto AWS .This application will host all our APIs that will be working on and requires some further setup like configuring the endpoints and the Server Endpoints and post this we can do a Sample deployment of a Hello World to have a sanity check of our endpoints and our overall Architecture

Deployment commands for the Deployment of App onto AWS

```
aws deploy create-deployment-group --application-name APP_NAME --
deployment-group-name APP_NAME_DepGroup --deployment-config-name
CodeDeployDefault.OneAtATime --ec2-tag-filters
Key=Name,Value=CodeDeploy,Type=KEY_AND_VALUE --service-role-arn
serviceRoleARN
```

Post the creation of a Deployment Group we can deploy it to our ec2 instance by using

```
aws deploy create-deployment --application-name APP_NAME_App --
deployment-config-name CodeDeployDefault.OneAtATime --deployment-
group-name APP_NAME_DepGroup --s3-location
bucket=codedeploydemobucket,bundleType=zip,key=APP_NAME _App.zip
```

1.5 Organization

The steps required for the project is:

Chapter 1

In this we provide a brief introduction to the project and the tools and Technologies Used

Chapter 2

In this chapter we did a literature survey of various sources to understand things like Spring, MVC Dependency Injection and AWS Services

Chapter 3

In this we go through with the High Level Generic Design that is being followed to build the API along with the Development Model Followed to create it

Chapter 4

In this Chapter we go through System Development in Brief and Look at Some High Level Components

Chapter 5

We decide on what the future work on this could look like

Chapter 2: Literature survey

2.1 Books and publication

To understand various things related to the Spring Framework I went through the following Resources which basically got a lot of things cleared up for me before I could move onto the implementation phase:

1) Spring Microservices in Action by John Carne

It got me to understand the very basics of different architecture types like monolithic architecture and microservices and helped me in understanding the differences between the two as well as pros and cons of each other. Then It also introduced to Spring Microservices that could be used

2) Amazon Web Services in Action

This resource helped me a lot in understanding various Amazon Web Service Resources that are available and how should we think about each one in designing our Services. Not only did it introduced us to various Services like Lambda and DynamoDB but also had in depth practical explanation of when should we be using those and also focused on reducing the costs for the Services that we use by Optimizing on our Resources

3) Java 9 Dependency Injection: Write Loosely Coupled Code with Spring 5 and Guice

This book introduced me to the fundamental concepts of Dependency Injection and had examples that can be easily related. Although the topic itself is very complicated to grasp in first go ,this book helped me understand it in and out with a large amount of Practical Examples

4) **RESTful Web APIs: Services for a Changing World**

This Helped me understand the Basics of a REST Service and best practices to follow while designing one.It has in depth explanation of what to do and what not to do and has helped me getting a strong grasp on the REST Principals

Chapter 3: System Development

3.1 SDLC Model Chosen

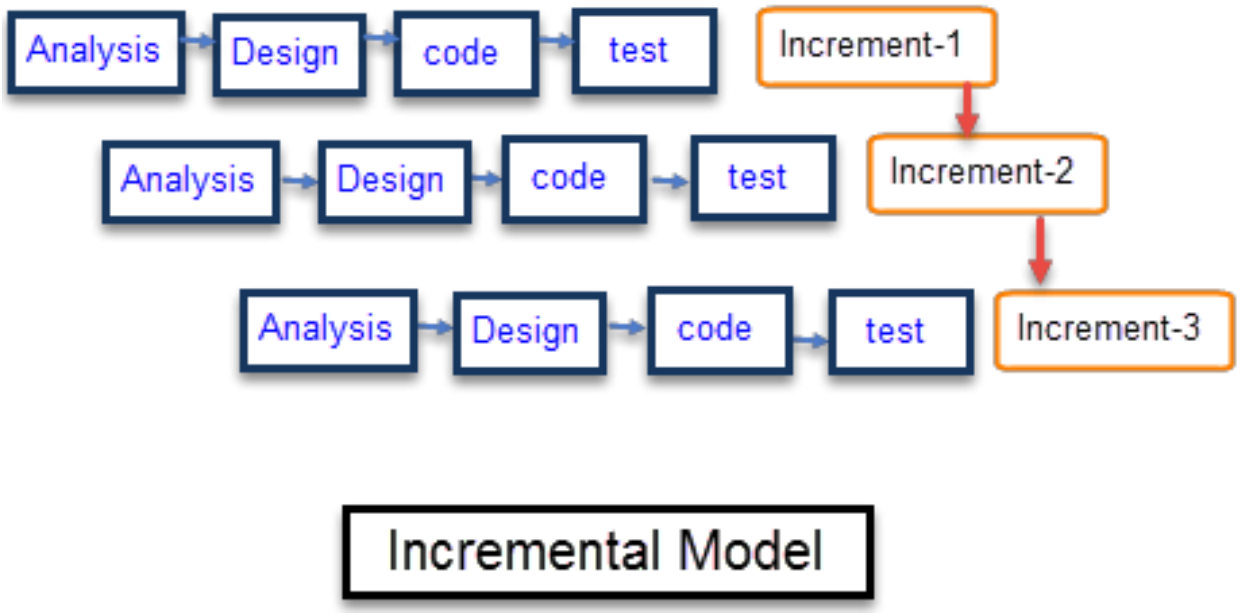

Fig 9:Incremental Model

We followed an Incremental Model while developing the feature.In an Incremental Model ,every time we do do 4 phases of Software Development that is Analysis,Design,Code and Test and after each Iteration whatever we release is known as an Increment.

Incremental Phases	Activities performed in incremental phases
Requirement Analysis	• Requirement and specification of the software are collected
Design	• Some high-end function are designed during this stage
Code	• Coding of software is done during this stage
Test	• Once the system is deployed, it goes through the testing phase

It can be thought of as breaking a major project into several deliverables and each increment is one such deliverable.In this Project Validating the Authentication header came as one of the maximum priority as without this we won't be able to provide access to any External Service.Not only this ,our API could have Been easily Misused if this feature was not present.So in the first Increment we created this specific module. Then I started working upon the Feature that can fetch the Balance and return the fetched balance to the required Service.This came of Second Priority as the Update Balance will also require this one post it has done the changes and updated the balance in the DB. And then in the third and Final Increment,We started working on Update Balance feature which takes in an input from the Service and updates the Balance in DB

Here are the Advantages and Disadvantages of using this SDLC model

Advantages	Disadvantages
• The software will be generated quickly during the software life cycle	• It requires a good planning designing
• It is flexible and less expensive to change requirements and scope	• Problems might cause due to system architecture as such not all requirements collected up front for the entire software lifecycle
• Throughout the development stages changes can be done	• Each iteration phase is rigid and does not overlap each other
• This model is less costly compared to others	• Rectifying a problem in one unit requires correction in all the units and consumes a lot of time

This model was chosen because the design for the API was already in place and minor changes that might occur were though of and accommodated in the design itself.

3.2 High Level Sequence Diagram for the API

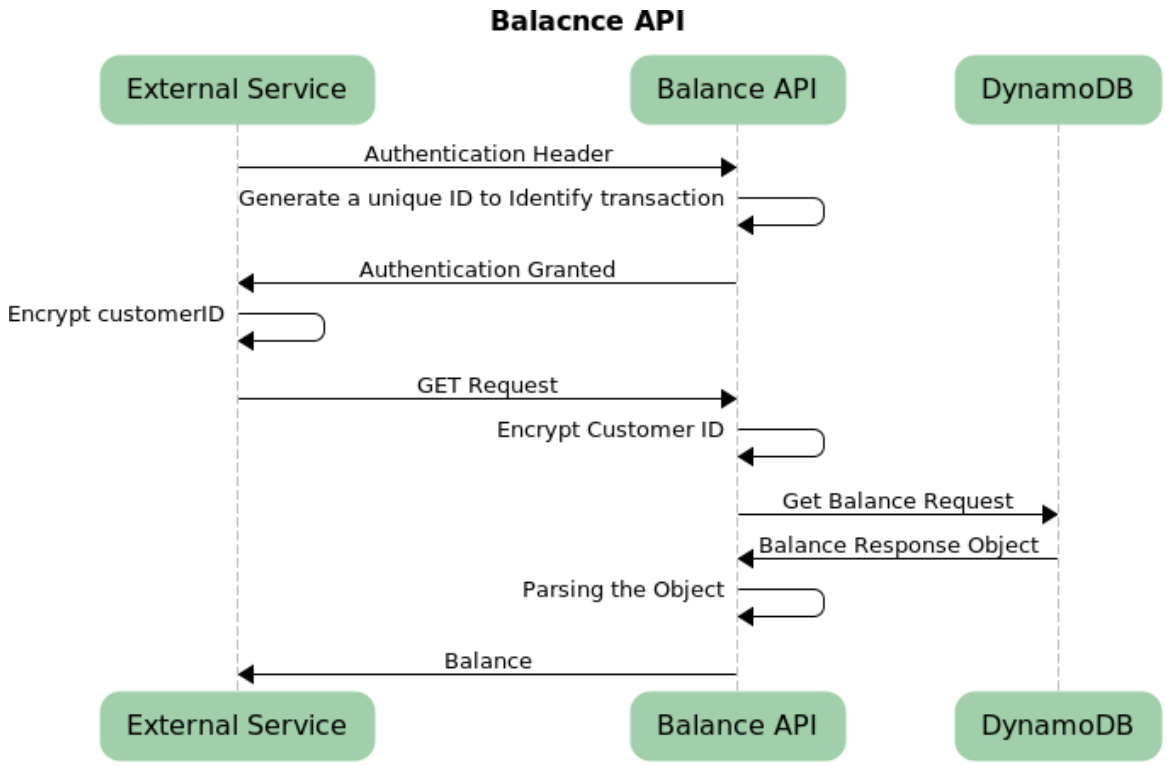

Fig: 9 GET Balance Sequence Diagram

There are mainly three major things that we should think of while designing our Service

1) External Service

These are the Services that will register to use our API, We don't want any unauthenticated service accessing this API in any way, but at the same time we want a mechanism in place that allows this API to be called from/integrated with any service that is registered to call it without much hassle and for that we come up with the Authentication Header. This Authentication Header mainly contains a uniquely generated String to identify the Service. When the Service first makes

this Request we first validate in our DB that if this particular ID is allowed to access the resources/operations that we have within the API

Without this authentication header we decide to terminate the Service and not go forward with any request from that ID. Post this validation we allow the Service to request our API with an encrypted customer ID

2) Balance API

Balance API consists of two major Operations

- Get Balance – Fetch the Balance of a Customer provided CustomerID
- Update balance – Update the Balance of a Customer Provided Amount

After we receive a customerID ,we first Unencrypt it and then encrypt it again by a different key value pair. Post this we convert our existing request to an Object Type and send that request to DynamoDB to Look up and Fetch the Balance

3) DynamoDB

DynamoDB is the Database that will eventually store the Information that we will need. It is highly secure and scalable for our task. At the same time it provides with very low read and write latencies and is therefore integrated with our API.

Update Balance Request Call

Update Balance request call again proceeds in a similar way to GET Balance Call. First the authentication header should be verified by our Service post which we need to send the External Service an response which Basically tells it if it has the

permission to proceed with the request. If it proceeds with the request without permission, its request will be Blocked

In case where an authentication is provided to the service, it makes a service Call to our Balance API, Sending it the amount and customerID ,which basically tells it to update the balance of customerId x to Balance Amount Rs .X. Our API listens for this and before proceeding with any other thing we encrypt the customer ID

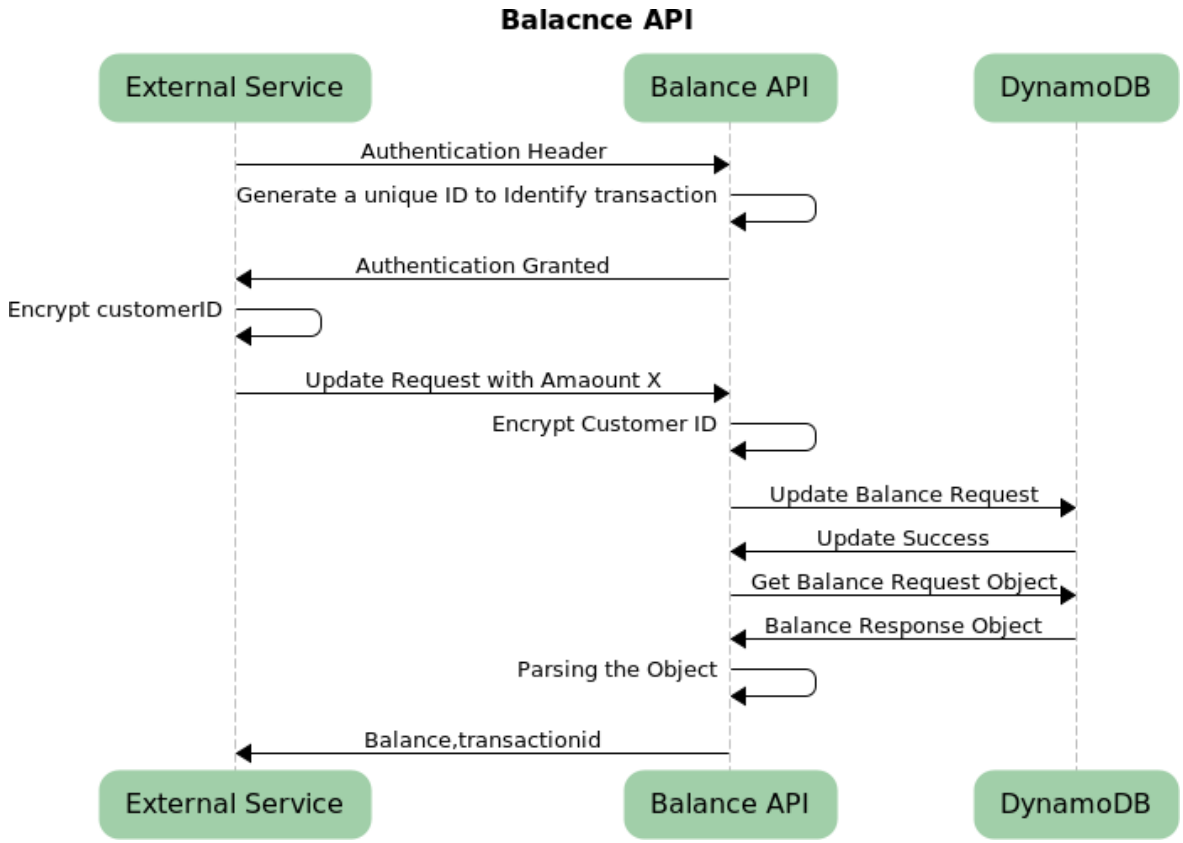

Fig 10:Update Balance Sequence Diagram

Post this we Do an Update Request on the Database and it returns us with a success or a failure Response .In Case of any Failure we invalidate the transaction and return nothing to the External Service and if it succeeds we update our API

with a success Response .After our API gets the Update Signal, it goes through with a get balance Request from the Backend Database. The DynamoDB in turn Responds with a Balance Object which is then Send back to The Balance API and json parsing of the object is done to fetch the balance from it and Return the calling Service an updated balance along with a transaction id that uniquely identifies the Transaction

Testing and Debugging

Unit tests for the following modules were written in Mockito

Mockito is based on JAVA and is usually used for the unit testing of a particular module.The Dependencies for that module can easily be mocked with some dummy functionality and tests can be written for the Module .

Chapter4: Performance Analysis

4.1 Evaluation:

The way that any Particular API could be Evaluated is dependent mainly on these basic factors:

- The Latency of the API/Service
- The Availability of the Service
- The Amount of traffic it can serve
- How easy is it to integrate with the Services that register to use it

For the Latency part ,we have a very low latency (in ms) because of the design that we chose and followed. The fact that we chose our architecture on the AWS

Stack made sure that it has very high availability and any downtime could be quickly recovered. It is also very scalable and while it can serve normal traffic with its current architecture designs, whenever there is a peak in traffic ,the Resources can be Scaled up to meet those Demands

Exact figures could not be added because they are confidential in nature

Chapter5: CONCLUSIONS

5.1 Conclusion:

In this project we have created a Balance API an deployed it onto an AWS Stack architecture. The API has a low latency, High availability and can be easily integrated with other Services

52. Future Scopes:

For the future scope we can try to add more functionalities to this API and bring down its latencies even further.

References

H. Y. Yang, E. Tempero and H. Melton, "An Empirical Study into Use of Dependency Injection in Java," *19th Australian Conference on Software Engineering (aswec 2008)*, Perth, WA, 2008, pp. 239-247, doi: 10.1109/ASWEC.2008.4483212.

S. Roubtsov, A. Serebrenik and M. van den Brand, "Detecting Modularity "Smells" in Dependencies Injected with Java Annotations," *2010 14th European Conference on Software Maintenance and Reengineering*, Madrid, 2010, pp. 244-247, doi: 10.1109/CSMR.2010.45.

S. Mostafa and X. Wang, "An Empirical Study on the Usage of Mocking Frameworks in Software Testing," *2014 14th International Conference on Quality Software*, Dallas, TX, 2014, pp. 127-132, doi: 10.1109/QSIC.2014.19.

H. Cinis, "Improving the Definition of Software Development Projects Through Design Thinking Led Collaboration Workshops," 2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), Gothenburg, 2018, pp. 254-255.

Ayush patel

ORIGINALITY REPORT

0%	0%	0%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.pinoy-web-application.com Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		