***MID TERM REPORT OF***

**APIs for Post Trade Investment Operation**

*A Graduate Project Report submitted to Manipal Academy of Higher Education in partial fulfilment of the requirement for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

**In**

**Electronics and Communication Engineering**

*Submitted by*

**Ashutosh Kumar Pandey**

**180907344**

*Under the guidance of*

**Pranita Hajare Rajiv Mohan David**

**HR & Assistant Professor**

**BlackRock, India Selection Grade**

**Department of ECE**

|  |
| --- |
| **A screenshot of a cell phone  Description generated with very high confidenceDEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**  MANIPAL-56104, KARNATAKA, INDIA |

**MARCH/APRIL 2022**

**ABSTRACT**

An API, or Application Programming interface, is a clearly defined set of methods used to access data. In essence, they allow applications to communicate with one another. A good API makes it easy to retrieve and manipulate data through a computer program by providing all the building blocks, which the programmer then uses to build the program. APIs provide database access to applications. An application makes a request for information via an API call. With the proper authorization, the application receives JSON data from another source it might not have otherwise have access to. JSON stands for JavaScript Object Notation and is a way of representing data. It’s lightweight, easy to read, and useful when handling data in languages like Python, Ruby, JS, etc.

The need for the automation of APIs for Post Trade Investment Operation is our users desire access to all of their data. In particular, clients desire access to their BlackRock data without having to use BlackRock proprietary applications. Users have several tools at their disposal to access this data, all of which have distinct use cases and best practices

.

APIs allow users to harness control of their data. Users can access data in proprietary applications built by their software development teams. Using APIs, users can make calls to BlackRock’s databases to retrieve and manipulate data. This empowers our users to build their own tools and fill gaps in their workflows not supported by native BlackRock application’s functionality. The project is built on various concepts including- Springboot, gRPC, API , database management and frontend-backend integration knowledge.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Contents** | | | | | |
|  | | | | | Page No |
| Abstract | | | |  | i |
|  | | | | | |
| **Chapter 1** | | | **INTRODUCTION** | |  |
|  | **1.1** | Introduction | | | iv |
|  | **1.2** | Motivation | | | vi |
|  | **1.3** | Organization of Report | | | vi |
|  | | | | | |
| **Chapter 2** | | | BACKGROUND THEORY | |  |
|  | **2.1** | Background Theory | | | vii |
|  | **2.2** | Current Workflow | | | xi |
|  | **2.3** | Proposed Structure | | | xii |
|  | | | | | |
| **Chapter 3** | | | **METHODOLOGY** | |  |
|  | **3.1** | Data Flow | | | xiii |
|  | **3.2** | Request and Response | | | xiv |
|  | **3.3** | Aladdin Graph | | | xiv |
|  | **3.4** | Protobufs | | | xv |
|  | | | | | |
| **Chapter 4** | | | **RESULT ANALYSIS** | |  |
|  | **4.1** | Result Analysis | | | xviii |
|  | **4.2** |  | | |  |
|  | | | | | |
| **Chapter 5** | | | **CONCLUSION AND FUTURE SCOPE** | |  |
|  | **5.1** | Work Conclusion | | |  |
|  | **5.2** | Future Scope of Work | | |  |
|  | | | | | |
| **REFERENCES** | | | | |  |
| **ANNEXURES (OPTIONAL)** | | | | |  |
| **PROJECT DETAILS** | | | | | xxii |

**CHAPTER 1**

**INTRODUCTION**

***1.1 Introduction***

In this chapter we shall discuss what are SWIFT Messages , the need for these messages within custodians and corporations and which aspect of portfolio management we shall be dealing with in this paper. We shall also discuss our motivation behind this project work and define our objective.

The increasingly global nature of business means that companies of all sizes often deal with multiple banking partners spread across the world. Typically,this implies integrating with a variety of bank communication interfaces and channels. As a result, treasurers and/or credit managers have to deal with significant operational burdens, often done manually to get an accurate and global view on liquidity and liability across their portfolio.

The various methods to interface with banks ranges from basic proprietary electronic banking applications, to customised host-to-host applications and, more recently, API integrations. As a business critical function, the security of a bank interface is of paramount importance for securing the associated data flows. The different interface options in the market place offer varied levels of security and reliability, which is a key consideration for any organization when

selecting which option to use.

By implementing SWIFT, Corporates benefit from a single highly secure and reliable window to communicate with all their banking partners using global standardised financial messages. This brings key benefits such as central visibility of cash positions, decreased operational risk and costs, and increased automation levels with integration possibilities to connect to a business’s existing systems.

SWIFT has been providing secure financial messaging to banks for over 40 years, and supporting more than 2,000 corporate groups by providing direct access to their network for 20 years. SWIFT’s reach is outstanding, connecting more than 11,000 banks and securities organisations across more than 200 countries and is well known for its resilience with 99,999% availability.

The Society for Worldwide Interbank Financial Telecommunications (SWIFT) technology is at the heart of most international money and security transactions. SWIFT is a large communications network that banks and other financial organisations use to send and receive information such as money transfer orders swiftly, precisely, and securely.

There are both operational and financial benefits for

corporates to join SWIFT, including:

• Simplification of communication by using a single window to connect to the banks with standardized messages that allows automation and integration with business applications.

• Improved cash visibility..

• Increased security.

• Scalability.

Behind most international money and security transfers is the Society for Worldwide Interbank Financial Telecommunications, known as the SWIFT system. SWIFT is a vast messaging network banks and other financial institutions use to quickly, accurately, and securely send and receive information, such as money transfer instructions.

An asset management company (AMC) is a firm that invests pooled funds from clients, putting the capital to work through different investments including stocks, bonds, real estate, master limited partnerships, and more. Along with high-net-worth individual (HNWI) portfolios, AMCs manage hedge funds and pension plans, and—to better serve smaller investors—create pooled structures such as mutual funds, index funds, or exchange-traded funds (ETFs), which they can manage in a single centralized portfolio.

AMCs are colloquially referred to as money managers or money management firms. Those that offer public mutual funds or ETFs are also known as investment companies or mutual fund companies. Such businesses include Vanguard Group, Fidelity Investments, T. Rowe Price, and many others.

AMCs are generally distinguished by their assets under management (AUM)—the amount of assets that they manage.

***1.2 Motivation***

In CSDR as a regulation coming into picture there is requirement in certain European markets that the trade has to settle on time. A trade that happens on a particular date suppose T has to settle on T+2 or T+3 depending on the market where the trade is getting executed. In most of the scenarios the trade gets settled within the stipulated timeframe, They may even settle in T+1 or T+3. In few cases they may fail to settle on the intended settle date .In these cases after CSDR coming into asset as a part of European market, the parties which are responsible for the trade to fail after intended settlement date would be given a debit penalty and then equivalent credit penalty would be given to counter party which failing because of the other counter party.

APIs are becoming part of BlackRock, and there is a renewed focus on enabling developers to write great, well-structured and well-documented APIs without needing a lot of time, effort or expertise. We need our APIs to adhere to a definition of types, and to not return a narrowed/flattened version of objects. This will help to empower innovation in BlackRock. As Settlement data forms a crucial part of the post trade lifecycle operations. The reference data scheme helps track real-time trade reconciliation status. This data is required across a number of processes related to accounting etc. and thus there is a need to provide a structured interface to access this data.

Our users desire access to all of their data. In particular, clients desire access to their BlackRock data without having to use BlackRock proprietary applications. Users have several tools at their disposal to access this data, all of which have distinct use cases and best practices. APIs allow users to harness control of their data. Users can access data in proprietary applications built by their software development teams. Using APIs, users can make calls to BlackRock’s databases to retrieve and manipulate data. This empowers our users to build their own tools and fill gaps in their workflows not supported by native BlackRock application’s functionality.

|  |  |
| --- | --- |
| *January 2022* | Compliance and basic courses |
| *February 2022* | AJDC Course Certification |
| *March 2022* | Project Architecture finalisation |
| *April 2022* | Proto files generation- backend development |
| *May 2022* | Frontend development |
| *June 2022* | Integration |

**Figure1**: Timeline for Project

***1.3 Organization Of Project Work***

The report is structured as follows. Chapter 1 gives an Introduction to the work, Motivation to do the project work and Organization.

Chapter 2 discusses the literature review and theoretical framework, Chapter 3 outlines the research design used and methodology adapted in the study.

Chapter 4 gives the result analysis and chapter 5 states the conclusion and future scope of work.

**CHAPTER 2**

**BACKGROUND THEORY**

In this chapter we shall give a brief background theory to the project work, current workflow and present the conceptual framework developed.

***2.1 Background Theory***

*SWIFT*

The Society for Worldwide Interbank Financial Telecommunication (SWIFT), formerly known as S.W.I.F.T. SC, is a Belgian cooperative society that provides services connected to the execution of financial transactions and payments between banks all over the globe. Its primary role is to act as the primary message network for initiating international payments. It also supplies financial institutions software and services, primarily for use on its proprietary "SWIFTNet" network and ISO 9362 Business Identifier Codes (BICs), sometimes known as "SWIFT codes."

SWIFT is a global electronic payment messaging system that is utilised by banks and financial organisations all over the world. Over 11,000 financial institutions from 200 countries utilise it. It is the communications system that allows financial institutions all around the world to exchange funds.

*SWIFT Code*

*Timeline

Description automatically generated*

**Figure2**: SWIFT Code

The Society for Worldwide Interbank Financial Telecommunication (SWIFT), formerly known as S.W.I.F.T. SC, is a Belgian cooperative society that provides services connected to the execution of financial transactions and payments between banks all over the globe. Its primary role is to act as the primary message network for initiating international payments. It also supplies financial institutions software and services, primarily for use on its proprietary "SWIFTNet" network and ISO 9362 Business Identifier Codes (BICs), sometimes known as "SWIFT codes."

Each Financial Institution has a unique 8 to 11-character code, which is also referred to as its bank identifier code (BIC) or ISO 93662 code.

Pre-Set Codes

* The Financial Institution (4 Characters)
* The Country Code (2 Characters)
* Location Code (2 Characters)
* Branch Code (3 Characters)

*Aladdin CSDR Penalties Overview*

The CSDR rules on the settlement discipline regime introduce several measures to prevent settlement fails by ensuring that all transaction details are provided to facilitate settlement, as well as further incentivizing timely settlement by cash penalty fines. In addition, CSDs are required to provide functionality to participants to ensure harmonization and automation of settlement processes across all European Economic Area markets to improve settlement efficiency.

CSDR Penalties are the most significant change to investment operations. Buy and sell side firms need to implement new processes and controls to manage the new operational risk of daily penalty charges. In response, Aladdin has implemented a new CSDR penalties workflow in settlements monitor based off the MT537 SWIFT message. The penalties workflow will help operations teams manage penalties from initial penalty receipt all the way through to cash posting and claim.

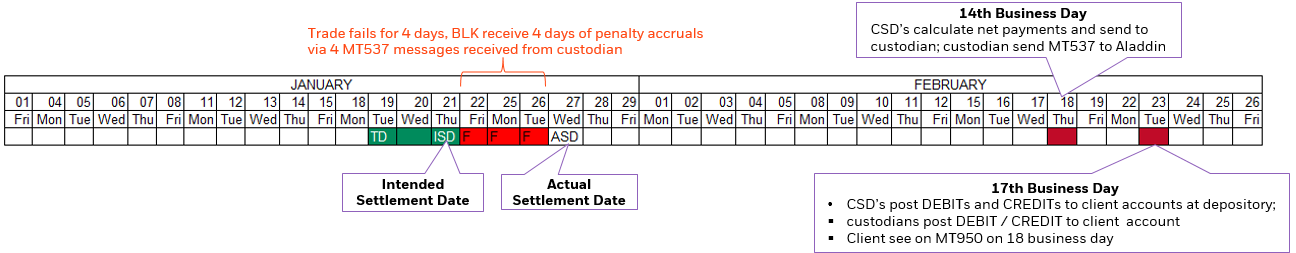
Before diving into the penalty workflow itself, it is important to understand how the CSDR penalty process will work. The key points to know are: -

* CSDR penalties are charged on every in-scope transaction for each day that a trade fails to settle starting with intended settlement date up until settle date or cancel date
* The industry best practice for communicating daily penalty charges is the MT537 SWIFT message.
* CSDs will advise the custodian on a daily basis via MT537
* The CSD will advise the custodian of the monthly net cash movement via MT537
* The custodian will send Aladdin daily MT537 messages and a monthly MT537 message
* The party responsible for the trade failure will be charged the penalty (debit) and the offended party will receive compensation for the fail (credit)
* The CSD for the trade attributes responsibility for the failed trade based on whichever party is last to the market or whichever party is short (assets or cash)
* The CSD will calculate and apply the penalty at the depository account level. The CSDs will not entertain challenges to this calculation unless there is a widespread system error
* The penalty calculation is based on the daily value of the failing portion of the trade
* The penalties are accrued on a daily basis but are not paid until the 17th business day of the following month; the depository will net the penalties at an account and broker level on the 14th business day of every month; the net values will be paid on 17th*Key*

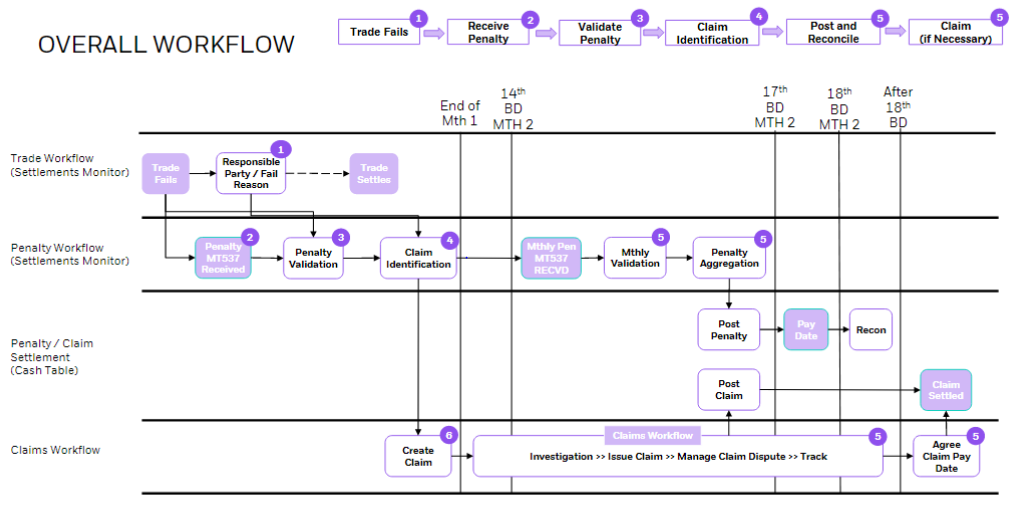
*Facts on Penalties:-*

* All in-scope trades accrue a debit or credit penalty from intended settlement date until actual settlement date or cancel date.
* The failing party receives a DEBIT and receiving party receives CREDIT; in essence it is a transfer of cash from one party to another.
* The depository simply attributes the DEBIT or CREDIT based on whichever account is last to market (late matching penalty ) or is SHORT assets or cash (late settlement penalty) or has placed the trade on HOLD (late settlement penalty ) .. The depository does not care about offline communication between the participants.
* The depository will calculate and apply the penalty at the depository account level; The penalty calculation is based on the daily value of the failing portion and the liquidity flag.
* The depository will not entertain challenges to the calculation unless there is a system error.
* The penalties are accrued on a daily basis but are not paid until the 17th business day of the following month ; the depository will net the penalties at an account and broker level on the 14th business day of every month; the net values will be paid on 17th.
* The depository will advise the custodian of the penalty accrual on a daily basis via MT537; the depository will advise the custodian of the monthly net via MT537.
* In the majority of cases, the custodian will send Aladdin a daily MT537 and monthly MT537.
* The depository will treat each failing trade independently and is unaware of the contingencies.

*Sample timeline for a single failed trade*

**

***2.2 Penalties Workflow***



**Figure4**: Current Workflow

* Processing of new SWIFT messages for daily penalty accruals (MT537)
* Processing of monthly aggregate MT537 messages
* Linking of penalties to their associated trade failure
* A "Penalties Tracking Workflow" within settlements monitor to track penalties from receipt on SD+1 through to settlement. This workflow will have new statuses specific to the penalties tracking workflow (more details below)
* Fail responsibility will be automatically applied on every trade fail in settlements monitor
* Users will be able to identify differences in "Fail Responsibility"
* QC validations will be performed on inbound MT537 messages to confirm the quantity matches our trade fail and that the price is within a tolerance.

**2.2.1 Daily Penalty Messages**

The daily penalty MT537 messages are intended to be a “pre-advice” of the penalty charge to inform market participants of upcoming cash movements in their account related to penalties on failed trades. As a “preadvice”, the industry standard practice dictates that no cash movements are to be created based on the daily penalty alone.

In Aladdin, the processing of daily penalties ensures you have timely transparency into potential charges to your clients and enables you to create controls that aid your firm in fulfilling its fiduciary duty. As you will see later in this document, all daily penalty messages with a matching tag 20 in Aladdin will be run through a series of automated quality control checks to give your operation scale while keeping tight control of your business.

**Daily Penalty Message Consumption**

Aladdin will consume and process the MT537 messages from custodians and store the data in Aladdin. For a SWIFT message to be successfully processed the minimum requirement is a valid fund code (custody account number) in Aladdin. This minimum requirement allows for use cases where your firm may want the penalty messages to be processed for trades done outside of Aladdin. For trades that contain a tag 20 on the SWIFT message that exists in your Aladdin, the common identifier can be used link the penalty to the failing trade in the new CSDR penalty workflows (QC checks, quick AladdinView lookups etc.). If an unsolicited penalty message is sent to your Aladdin environment that does not contain the minimum required data to be processed the message will be discarded.

**Viewing Daily Penalties**

All daily penalty data will be available in AladdinView in the new “Penalties” tab. AladdinView will be able to retrieve penalties no matter what the status or state of the penalty. From AladdinView, you will be able to edit the daily penalty as part of exception management in the daily penalty quality control workflow (more detail found later in this document). Penalties that fail to pass quality control checks will be viewable in Dashboard

**2.2.2 Monthly Penalty Messages**

The monthly MT537 penalty message will contain the aggregate net penalty amount at the account + broker + currency level. Before we dive into how Aladdin will process the monthly penalty message, there are key facts you should know about how this message will be used across the industry:

* This message will be received on the 14th business day of each month for the penalty payments associated with trades that settled in the prior month. The payments will settle on the 17th business day.
* If a trade fails over a month-end, all penalties associated with that trade will be paid in the month following actual settlement date.
* MT537 messages are the golden source for the expected cash movements driven by penalties

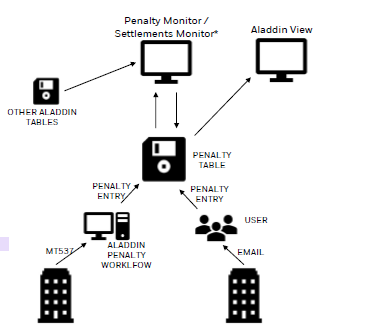
**Monthly Penalty Message Processing & Consumption**

Monthly MT537 messages will be consumed by Aladdin and automatically generate the cashflows in Aladdin for the credits and debits in your client’s account. While the custodian is providing a net penalty amount, the SWIFT message contains trade and penalty level details which Aladdin will use to post the cash at the trade level.

As stated previously, the MT537 is the golden source for the expected penalty cash flows. To ensure there are no breaks, Aladdin will generate cash postings for penalties using this monthly message. Aladdin will generate MISC CASH postings by aggregating the penalty charges for each trade using the PCOM references (unique id for daily penalties) contained in the monthly MT537 message and posting at the trade level. To identify the differences between the daily and monthly messages, the contents of the monthly MT537 message will be compared against the aggregated daily penalty messages and generate offset daily penalty records with a penalty type of “Penalty Difference” (PNDF) to easily identify where the monthly and daily amounts did not match. To facilitate efficient matching in the cash reconciliation process, both the monthly MT537 and the MT950 will contain a shared “CPRF” ID that Aladdin will use to match off the cash movements.

Penalty API primarily focus upon GET API (given the certain details like fund, investment number we are able to get associated penalties) .In Aladdin we have just one fail record for any given trade and that record keeps on getting updated as the trade continues in this settlement cycle. But there may be multiple penalties associated with one transaction because penalties are calculated at the EOD for all the days where the trade is failing

***2.3 Proposed Structure***



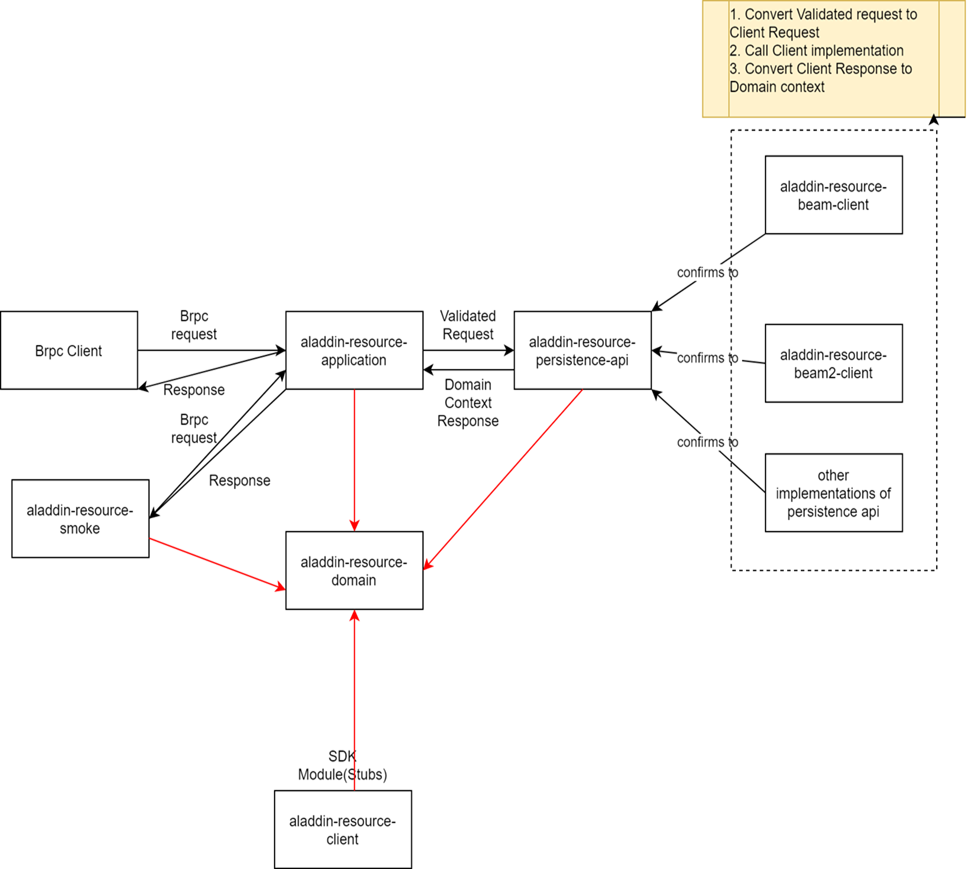
**Figure5**: Proposed Application Architecture.

**CHAPTER 3**

**METHODOLOGY**

In this chapter we shall give an introduction to the Architecture and methodology of the project.

***3.1 Microservice Architecture***



**Figure6**: Microservice Architecture.

***3.2 Request and Response***

## Penalty APIs –

In Penalty Service we would be exposing bRPC APIs to access and mutate the Penalty Data.

We can target the API development in 2 phases for read and create/update APIs.

* **Read Penalty**

Method: GET

Query Parameters:

* + fund & investment number
  + fund, investment number & penalty date range
  + internal id
  + internal id & penalty date range
  + settle date range (to/from) : TBD

Output: List of Penalties

* **Create/Update Penalty**

Method: POST

* + For Create and Update Penalty APIs we would take reference from Aladdin View where we support selective create/update of fields.
  + **Create**: To create a new penalty the model should contain the information for the following fields:

Fund, Invnum, Int Id, Penalty Date, Status Code, Penalty Amount, Currency, Penalty Reference, Penalty Status, Reason Code, Status message.

* + **Update**: The update would only be supported for the following fields:

Status code, penalty amount, penalty status, reason code, status message.

***3.3 Aladdin Graph***

APIs are becoming part of Aladdin Studio, and there is a renewed focus on enabling developers to write great, well- structured and well-documented APIs without needing a lot of time, effort or expertise.At the moment, there's a lack of consistency in our APIs. While field names are the same, APls may remove fields or even flatten nested fields as they see fit.

This means that two APIs that should return the same object end up returning two different shapes - as a user, you can imagine how frustrating this might be!

We need our APIs to adhere to a definition of types, and to not return a narrowed/flattened version of objects. This will help to empower innovation in and on Aladdin OS (in: Aladdin Developer on: Aladdin Studio users).

*Aladdin Graph Structure*

* one and only one graph
* not all the segments of the graph are connected to each other

graph is defined using labels

* message and apis are tagged with labels

*Aladdin Graph Definition*

* Entities/Nodes are defined as messages.
* Entities are defined using protocol-buffer.
* APIs are defined as services.

As part of the Aladdin Graph pipeline we:

* Lint our protos to ensure it conforms to our set of guidelines.
* Generate code from our protos:

bRPC Java API service & client stubs. bRPC (Blackrock Remote Procedure Call) is the interface between google protobuf and bms, based on gRPC.

* BMS GoLang API service & client stubs.
* OpenAPI2.0 conformant Swagger specifications.
* bRPC Gateway plugins. bRPC Gateway is a gateway for exposing our BMS services over HTTP externally and internally.
* Deploy the code generated from our protos.

***3.4 Protobufs***

Protocol buffers are Google's language-agnostic, platform-agnostic, flexible serialisation technique — imagine XML, but smaller, quicker, and simpler. Once you've defined how our data should be formed, you may utilise custom generated source code to simply write and read structured data to and from a number of data streams and languages.

The application we'll use as an example is a simple "address book" that can read and write people's contact information to and from a file. Each contact in the address book contains a name, identification number, email address, and phone number.

How do we serialize and retrieve structured data like this?

With protocol buffers, you write a .proto description of the data structure we wish to store. From that, the protocol buffer compiler creates a class that implements automatic encoding and parsing of the protocol buffer data with an efficient binary format. The generated class provides getters and setters for the fields that make up a protocol buffer and takes care of the details of reading and writing the protocol buffer as a unit. Importantly, the protocol buffer format supports the idea of extending the format over time in such a way that the code can still read data encoded with the old format.

To create our address book application, we'll need to start with a .proto file. The definitions in a .proto file are simple: we add a message for each data structure we want to serialize, then specify a name and a type for each field in the message. Here is the .proto file that definesour messages, addressbook.proto.

Text

Description automatically generated

The .proto file starts with a package declaration, which helps to prevent naming conflicts between different projects. In Java, the package name is used as the Java package unless we have explicitly specified a java\_package, as we have here. Even if you do provide a java\_package, we should still define a normal package as well to avoid name collisions in the Protocol Buffers name space as well as in non-Java languages.

After the package declaration, we can see three options that are Java-specific: java\_multiple\_files, java\_package, and java\_outer\_classname. java\_package specifies in what Java package nameour generated classes should live. If we don't specify this explicitly, it simply matches the package name given by the package declaration, but these names usually aren't appropriate Java package names (since they usually don't start with a domain name). The java\_outer\_classname option defines the class name of the wrapper class which will represent this file. If we don't give a java\_outer\_classname explicitly, it will be generated by converting the file name to upper camel case. For example, "my\_proto.proto" would, by default, use "MyProto" as the wrapper class name. The java\_multiple\_files = true option enables generating a separate .java file for each generated class (instead of the legacy behavior of generating a single .java file for the wrapper class, using the wrapper class as an outer class, and nesting all the other classes inside the wrapper class).

Next, we haveour message definitions. A message is just an aggregate containing a set of typed fields. Many standard simple data types are available as field types, including bool, int32, float, double, and string. We can also add further structure to our messages by using other message types as field types – in the above example the Person message contains PhoneNumber messages, while the AddressBook message contains Person messages. We can even define message types nested inside other messages – as we can see, the PhoneNumber type is defined inside Person. We can also define enum types if we want one of our fields to have one of a predefined list of values – here we want to specify that a phone number can be one of the following phone types: MOBILE, HOME, or WORK.

The " = 1", " = 2" markers on each element identify the unique "tag" that field uses in the binary encoding. Tag numbers 1-15 require one less byte to encode than higher numbers, so as an optimization we can decide to use those tags for the commonly used or repeated elements, leaving tags 16 and higher for less-commonly used optional elements. Each element in a repeated field requires re-encoding the tag number, so repeated fields are particularly good candidates for this optimization.

Each field must be annotated with one of the following modifiers:

optional: the field may or may not be set. If an optional field value isn't set, a default value is used. For simple types, we can specify our own default value, as we've done for the phone number type in the example. Otherwise, a system default is used: zero for numeric types, the empty string for strings, false for bools. For embedded messages, the default value is always the "default instance" or "prototype" of the message, which has none of its fields set. Calling the accessor to get the value of an optional (or required) field which has not been explicitly set always returns that field's default value.

repeated: the field may be repeated any number of times (including zero). The order of the repeated values will be preserved in the protocol buffer. Think of repeated fields as dynamically sized arrays.

required: a value for the field must be provided, otherwise the message will be considered "uninitialized". Trying to build an uninitialized message will throw a RuntimeException. Parsing an uninitialized message will throw an IOException. Other than this, a required field behaves exactly like an optional field.

**CHAPTER 4**

**RESULT ANALYSIS**

In this chapter we shall look at some of the generated code and see what classes and methods the compiler has created for us.

For each field in the message, both messages and builders have auto-generated accessor methods; messages only have getters, while builders have both getters and setters. The accessors for the Person class are listed below (implementations omitted for brevity)

Text

Description automatically generated

Meanwhile, Person.Builder has the same getters plus setters:

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

As we can see, there are simple JavaBeans-style getters and setters for each field. There are also has getters for each singular field which return true if that field has been set. Finally, each field has a clear method that un-sets the field back to its empty state.

Repeated fields have some extra methods – a Count method (which is just shorthand for the list's size), getters and setters which get or set a specific element of the list by index, an add method which appends a new element to the list, and an addAll method which adds an entire container full of elements to the list.

These accessor methods use camel-case naming, even though the .proto file uses lowercase-with-underscores. This transformation is done automatically by the protocol buffer compiler so that the generated classes match standard Java style conventions.

**CHAPTER 5**

**CONCLUSION AND FUTURE SCOPE OF WORK**

Implementation of backend using SpringBoot

**REFERENCES**

1. Developers.google.com
2. Investopidea-SWIFT
3. Blackrock wiki

PROJECT DETAILS

|  |  |  |  |
| --- | --- | --- | --- |
| *Student Details* | | | |
| **Student Name** | **Shladhya Shiti** | | |
| Register Number | 180907538 | Section / Roll No | A 40 |
| Email Address | Shladhya.s@gmail.com | Phone No (M) | 8877222333 |
| **Student Name** | **NA** | | |
| Register Number |  | Section / Roll No |  |
| Email Address |  | Phone No (M) |  |
|  | | | |
| *Project Details* | | | |
| **Project Title** | **SWIFT MT950 Display** | | |
| Project Duration | 6 months | Date of reporting | 17th January’22 |
| Expected date of completion of project | 30th June’22 |  |  |
|  |  | | |
| *Organization Details* | | | |
| **Organization Name** | **Blackrock India** | | |
| Full postal address with pin code | 14th & 15th Floor, Tower C & D,DLF Building No. 14, DLF Cyber City Phase-III, Gurgaon, Gurgaon, INDIA, 122002. | | |
| Website address | https://www.blackrock.com | | |
|  |  | | |
| *Supervisor Details* | | | |
| **Supervisor Name** | **Pranita Hajare** | | |
| Designation | HR | | |
| Full contact address with pin code | 14th & 15th Floor, Tower C & D,DLF Building No. 14, DLF Cyber City Phase-III, Gurgaon, Gurgaon, INDIA, 122002. | | |
| Email address | Pranita.Hajare@blackrock.com | Phone No (M) |  |
|  |  | | |
| *Internal Guide Details* | | | |
| **Faculty Name** | **Akshatha K.R** | | |
| Full contact address with pin code | Dept. of E&C Engg., Manipal Institute of Technology, Manipal – 576 104 (Karnataka State), INDIA | | |
| Email address | akshatha.kr@manipal.edu | | |