Project #: MXC-ABVO


WPI

BNP PARIBAS

# BNP Paribas: Symphony Chatbot

A Major Qualifying Project Report

*Submitted to the Faculty of*
*Worcester Polytechnic Institute*
*In Partial Fulfillment of the Requirements for*
*the Degree of Bachelor of Science*

Submitted By:
**Matthew Nicholson**, Management Engineering
**Yaofeng Wang**, Computer Science and Industrial Engineering
**Yiyi Chen**, Industrial Engineering

Submitted to:
*On-site Liaison:*
**Andrew Clark**

*Project Advisors*:
**Professor Michael Ciaraldi**, Department of Computer Science
**Professor Renata Konrad**, Foisie School of Business
**Professor Kevin Sweeney**, Foisie School of Business

Submitted on:
January 23, 2019

# Abstract

To reduce the time and effort required to complete locate short sale requests, our team automated the communication process between the equity and derivative traders (EQD) and the stock lending and borrowing traders (SLAB) at BNP Paribas. Currently, for EQD traders to perform a short sale, they must contact the SLAB traders to figure out if they are allowed to borrow the shares. Once approved, the SLAB trader will be able to fill out a form and submit it to the SLAB software for further approval and borrowing rates. The SLAB traders review the request and reply back via Symphony chat room to let the EQD trader know if they have been approved, declined, or partially approved. The SLAB trader will also provide the EQD trader with information regarding the rate at which the shares are being borrowed. The current communication process is complex with multiple unnecessary manual steps. The chatbot our team designed will speed up the communication process by automating all the manual steps required by SLAB. The chatbot will also notify EQD traders of trades at a faster rate since it sends out responses automatically. Ultimately, we have created a chatbot that will act as “middle man” between the two traders to accelerate and simplify the current process.

## Acknowledgements

We would like acknowledge everyone who lent a helping hand to us during our MQP. We would like to especially thank the following people at BNP Paribas who guided us during our eight weeks on-site and made the success of our project possible:

Andrew Clark
Arnab Mukherjee
Jacob Brooks
Maxime Bourgeon
Kumar Samy
Steve Szymanski
Nicolas Wright
Sundar Govindasamy
Ziad Iskander
Julie Clark
Cecile Malinaud
Brian Cahalan
William Hau

We would also like to say thank you to the faculty at WPI who selected us for this project, helped us, supplied us with resources, and most importantly pushed us to be the most successful MQP group that we could be:

Kevin Sweeney
Michael Ciaraldi
Renata Konrad
Jon Abraham
Wilson Wong

## Executive Summary

Every day, equity and derivative (EQD) traders at BNP communicate with stock lending and borrowing (SLAB) traders to try and execute short sale locates. It is the job of the SLAB traders to process all of these requests and provide a decision based on what shares are available. Specifically, the decision is based on how many shares are available to be borrowed and then determine the rate at which those shares can be borrowed. The current communication process between these two divisions is slow and outdated. We have been tasked with creating a chatbot that will automate this communication process. The chatbot will act as a "middle man" between the two traders.

Currently, for an equity trader to communicate with the SLAB desk they must find an available SLAB trader through Symphony, which is a chat platform, and ask if they are allowed to make a trade. Once the permission is confirmed SLAB traders then create a proxy, which is a server that acts as an intermediary for requests from clients seeking resources from other servers. After creating the proxy, the trader can submit and monitor the requests made by the EQD trader in the Falcon software. SLAB traders can check their SSA (security and safety administration) monitor to see pending requests within Falcon. After the request has been approved it will pop up in Falcon. From there, the SLAB trader can forward this message to the EQD trader for them to accept or decline the offer.

Our chatbot will run a similar process but will eliminate the need to use multiple programs to make a locate request. Traders no longer have to create a proxy, communicate with Falcon, or check the SSA monitor. Our chatbot does all the communication; all the trader has to do is submit a request and our chatbot will communicate with SLAB. Our chatbot is able to read locate requests from an EQD trader, such as "I would like to short 100 share's SPY" that message is then forwarded to the SLAB desk where those traders will determine if the locate request should be approved, declined, or partially approved using their Falcon software. The chatbot will then forward the approval message to back to trader. After the approval has been sent back to the equity trader he/she will have the option to accept or decline the approved locate request. Regardless of the response from the EQD trader, the chatbot will receive and send the response back to the SLAB desk for their records, thus completely eliminating any direct communication between the two parties.

Our top priority was to code the main function of the chatbot so that the secondary methods, which were also coded by our team, can communicate seamlessly with the API's we were working with. Not only does the chatbot need to communicate with EQD and SLAB traders, but also with a REST API, SOAP API/Falcon. The chatbot's ability to communicate with users and software's eliminates the manual steps that were originally being used to complete short sales.

We conducted significant data and system analyses and we hope that the progress we've made with our designs and implementation will simplify the process and significantly reduce time spent executing short sale locates.

# Authorship

| Section | Author |
|---|---|
| Abstract | Matthew Nicholson |
| Acknowledgements | Matthew Nicholson |
| Executive Summary | Matthew Nicholson |
| Authorship | Yaofeng Wang |
| Table of Contents | Yaofeng Wang |
| Chapter 1: Introduction | Matthew Nicholson |
| 1.1 BNP Paribas Background | Matthew Nicholson |
| Chapter 2: Background | |
| 2.1 Symphony Chat Platform | Matthew Nicholson |
| 2.2 Examples of Industry Implementation | Yiyi Chen |
| 2.3 Short Sale Locate Request | Matthew Nicholson |
| 2.4 Current Communication | Matthew Nicholson |
| 2.5 Advantages and Disadvantages | Matthew Nicholson |
| 2.5.1 Advantages | Matthew Nicholson |
| 2.5.1.1 Data Flow | Matthew Nicholson |
| 2.5.1.2 Acceptance and Declines | Matthew Nicholson |
| 2.5.2 Disadvantages of Current System | Matthew Nicholson |
| 2.5.2.1 Not Completely Automated through Symphony | Matthew Nicholson |
| 2.5.2.2 Process Speed | Matthew Nicholson |
| Chapter 3: Methodology | |
| 3.1 Problem Statement | Matthew Nicholson |
| 3.2 Objectives | Matthew Nicholson |
| 3.2.1 Deliverable 1 | Matthew Nicholson |
| 3.2.2 Deliverable 2 | Matthew Nicholson |
| 3.3 Technology Tools | Yaofeng Wang |
| 3.3.1 Programming Language and IDE | Yaofeng Wang |
| 3.3.1.1 Python | Yaofeng Wang |
| 3.3.1.2 PyCharm | Yaofeng Wang |
| 3.3.2 Prompts and Commands | Yaofeng Wang |
| 3.3.2.1 Git Command | Yaofeng Wang |
| 3.3.2.2 Conda and Anaconda Prompt | Yaofeng Wang |
| 3.3.3 Version Control Systems | Yaofeng Wang |
| 3.3.3.1 BitBucket | Yaofeng Wang |
| 3.3.3.2 JFrog Artifactory | Yaofeng Wang |
| 3.3.4 Application Programming Interface (API) | Yaofeng Wang |
| 3.3.4.1 Symphony API | Yaofeng Wang |
| 3.3.4.2 SOAP/Falcon API | Yaofeng Wang |
| Chapter 4: Process Analysis | |
| 4.1 Architectural Overview | Yiyi Chen |
| 4.1.1 Use Case Diagram | Yiyi Chen |
| 4.1.2 Swim-Lane Diagram | Yiyi Chen |
| 4.1.3 Data Flow Diagram | Yiyi Chen |
| 4.2 Chatbot Process Design | Yiyi Chen |
| 4.2.1 Flowchart | Yiyi Chen |
| Chapter 5: System Analysis | |

| | |
|---|---|
| 5.1 Message Processing | Yaofeng Wang |
| 5.1.1 Language Processing | Yaofeng Wang |
| 5.1.1.1 Hi | Yaofeng Wang |
| 5.1.1.2 Method | Yaofeng Wang |
| 5.1.1.3 Help | Yaofeng Wang |
| 5.1.1.4 Bye | Yaofeng Wang |
| 5.1.2 Using the Methods to Short a Stock | Yaofeng Wang |
| 5.1.3 Status | Yaofeng Wang |
| 5.1.3.1 Status RequestID | Yaofeng Wang |
| 5.1.3.2 Status All | Yaofeng Wang |
| 5.1.4 “I do not Understand” | Yaofeng Wang |
| 5.2 Error Handling | Yaofeng Wang |
| 5.2.1 Invalid Single Request | Yaofeng Wang |
| 5.2.2 Invalid Multiple Request | Yaofeng Wang |
| 5.2.3 Invalid Status Request | Yaofeng Wang |
| 5.2.4 Invalid Commands | Yaofeng Wang |
| 5.2.5 Server Error | Yaofeng Wang |
| Chapter 6: Results | |
| 6.1 Functional Chatbot | Matthew Nicholson |
| 6.2 User Friendly | Matthew Nicholson |
| 6.3 Special Features | Matthew Nicholson |
| Chapter 7: Recommendations | |
| 7.1 Testing | Yiyi Chen |
| 7.2 Falcon | Yiyi Chen |
| 7.3 Chatbot | Yiyi Chen |
| Chapter 8: Conclusion | Matthew Nicholson |
| Chapter 9: Reflection on the project | |
| 9.1 Discussion of design in the context of the project | Yiyi Chen |
| 9.2 Discussion of constraints considered in the design | Yiyi Chen |
| 9.3 Discussion of the need for life-long learning | Yiyi Chen |
| Bibliography | All |

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1: Introduction

BNP Paribas, is an international banking group. Their slogan is "The bank for a changing world". This is exemplified by our project to automate the manual steps of locate approval process between equity traders and the stock lending and borrowing traders. With guidance from multiple departments across the Global Markets (GM) IT, our team was given the opportunity to develop a chatbot on Symphony that will automate the entire process. By simplifying and automating the communication process we will have created an easier way for traders to execute short sales. As a result, BNP equity and SLAB traders can focus on other tasks within BNPP to improve the banks overall performance.

## 1.1 BNP Paribas Background

Banque Nationale de Paris (BNP) Paribas began in 1848, when its two companies, Comptoir National d'Escompte de Paris (CNEP) and Comptoir National d'Escompte deMulhouse, were established. These "comptoirs d'escompte" (discounting houses) were formed to facilitate credit circuits in France while the country was undergoing an economic meltdown and political revolution which had destroyed the country's former credit system. In 1872, European bankers wanted to "raise funds to borrow to free up regions and, on a longer term, to acquire shareholdings in companies and acquire a stake on capital markets (BNP History);" thus, Banque de Paris et des Pays-Bas (Paribas) was established (History of the bank over the last two centuries, n.d.).

In 2000 a merger of Banque Nationale de Paris and Paribas led to the creation of BNP Paribas.

BNP Paribas is the largest French banking group and the largest bank in the Eurozone. It became one of the five largest banks in the world following the 2008 financial crisis. Despite some legal difficulties in the United States in 2014, they've managed to stay as one of the largest banks in the world.

Today, BNP Paribas has a presence in over 70 countries on 5 continents (BNP History;) and employs approximately 189,000 employees as of 2015. BNP Paribas's key activities include retail banking (i.e., corporate vehicle leasing, rental and financial solutions, and online savings and brokerage) and financial services (i.e., private banking, asset management, and real estate services) as well as corporate and institutional banking (e.g., solutions across capital markets, securities services, financing, treasury, and financial advisory).

BNP Paribas recognizes that it is a changing world, and encouraging innovation is the firm's response to better serve their customers (History of the Group. n.d.). The organization prides itself on their diversity and forward thinking and they have been rewarded for it as well. BNP has been awarded as the most diversified banks sector in Europe for Vigeo-Eiris in 2017, Most Innovative Investment Bank for Climate Change and Sustainability 2017 by The Banker Magazine, and Green Bond Lead Manager 2018 by Environmental Finance.

# Chapter 2: Background

## 2.1 Symphony Chat Platform

Bloomberg financial software provides real-time information of all the world markets, and allows instant messaging through applications online and mobile devices. This feature enables Bloomberg to dominate in the financial sector. However, in 2014, Bloomberg reporters were accused of prying activity of terminal users (Symphony 2018). This act led financial institutions to invest in something new and more secure.

Symphony was that investment. Symphony is a secure, cloud-based, communication and content sharing platform. Symphony enables businesses to provide best-of-breed collaboration technology to their employees, while driving efficiency and ensuring security and compliance. Enterprise customers can now centralize all workflow in a single platform, using Symphony as a replacement for traditional email and voice systems to securely communicate with internal and external teams, share documents and content, and conduct meetings with conferencing and screen-sharing. Symphony's success is a direct result of its powerful platform, customizable user experience, and open partner ecosystem which delivers a large and growing market of applications and integrations (Symphony Blog, n.d.).

The Symphony technology was first built as an internal messaging system by Goldman Sachs called Live Current. In October 2014, Goldman Sachs along with 14 other financial institutions created and invested $66M USD in Symphony Communication Services LLC and acquired Perzo, Inc., a secure communication application that provided end to end encryption messaging (Symphony, 2018).

Perzo was founded by David Gurle in 2012 and David is currently the company's CEO. He was involved in developing the communication offerings at Skype, Thomson Reuters, and Microsoft.

On September 15, 2015, Symphony made a public release of its platform and announced partnerships with DowJones, McGraw Hill Financial, and Selerity. McGraw Hill Financial will integrate its financial information tool, S&P Capital IQ and Dow Jones will provide its entire live news feed of about 10,000 stories to the new platform. Selerity will deliver contextually relevant news, research and their proprietary breaking news notifications, directly into the Symphony platform (Symphony Blog, n.d.).

In May 2017, Symphony raised $63 M USD in additional funding from BNP Paribas as well as its existing investors, bringing the total valuation above $1 B USD. Symphony previously raised a total of $170 M USD from the world's leading financial firms, venture capital firms, and Google.

"Digital transformation is central to BNP Paribas Global Markets' strategy, and collaboration with new financial technology as a crucial part of that process. Forming agile partnerships with exciting and innovative companies like Symphony helps us deliver an exceptional service to clients and remain their partner of choice in a changing world," said Olivier Osty, Executive Head of Global Markets, BNP Paribas. Symphony (2018)

## 2.2 Examples of Industry Implementation

An application programming interface (API) enables Symphony to communicate among various components, which means data from other systems is available in Symphony. Thus, programmers can code while using open-source applications. An API accelerates cross functioning on the Symphony platform by connecting to chatbots and other softwares throughout any company. Therefore, it is quick and easy to build chatbots that can connect to other client ecosystem components and trading platforms. For instance, Natixis, a French investment bank in Asia, replaced twenty applications with one Symphony chatbot that brought all the applications together. The chatbot also allowed the bank to send messages automatically, ensuring the client would receive their messages immediately. In other words, the salesperson could examine a client's trading status in a single step. Frédéric Dalibard, a banker at Natixis, said trading time was cut down from 15 minutes to 20 seconds (Bartholomew, 2018).

Furthermore, Symphony chat is far more convenient than email and more business related than other instant chat platforms such as Skype. The faster-messaging speed removes time wasted and reduces operational wastes. Moreover, chatbot implementation alongside Symphony has been correlated with generating more revenue.

With increased popularity surrounding Symphony other banks are beginning to join the wave of building Symphony chatbots. More and more, banks are working on creating their own workflow chatbots on Symphony. Nomura, which has 13,000 licensed individuals, has come up with 1000 chatbot ideas, from complex analytical tools to meeting room booking tasks (Bartholomew, 2018). BNP Paribas invested in Symphony during 2017 and is currently using Symphony both internally and externally with the hope of interacting with more customers. BNPP offices in Hong Kong, Singapore, Tokyo, and other offices have built some chatbots for trading (Kilpatrick, 2018).

JP Morgan also has approximately twenty-five Symphony projects in progress. JP Morgan has constructed a chatbot that collects relevant information when a product is discussed and pops the price back automatically to the salesperson (Quan, 2018). Currently, BlackRock is using Symphony for internal and external messaging (Norton, 2017).

To summarize, Symphony is a new open-source platform, which allows instant messaging and customized chatbots. The one-stop feature eliminates the operational cost and promotes trading speed. The number of Symphony users reached more than 300,000 in 2018 (Bartholomew, 2018).

## 2.3 Short Sale Locate Request

A locate request is a message sent to the SLAB desk that helps execute short sales. A short sale is the sale of an asset (securities or other financial instrument) that the seller does not own. The seller affects such a sale by borrowing the assets to deliver it to the buyer. To execute a short sale an equity trader needs to borrow shares that are held within the market. To get these borrowed shares equity traders must communicate with the SLAB desk because they manage all the shares available for borrowing. SLAB traders use a database that lists the number of shares

available for borrowing when determining their decision. When a request comes in from a trader, SLAB processes the request, determines if the shares are available, and then reports back to the trader whether the request has been approved, partially approved, or declined. They will also give the trader the rate at which the shares will be borrowed.

The rates change depending on if the request is easy or hard to borrow. The harder it is to find shares in the market the higher the rate will be to borrow them. For example, Tesla's stock price is considered to be overvalued by many analysts. Tesla is seen in this way because of a considerably high stock price (302.26) for a company with a turbulent CEO and large amounts of debt. This has caused traders to think Tesla stock will fall dramatically in the near future. Do to this feeling among investors many traders have been attempting to execute short sales on Tesla. Whenever a large number of traders are taking the same position (short or long) there will be a shortage of shares in the marketplace. Investors believe Tesla stock is overvalued and will fall soon, thus creating a shortage due to the high volume of shorts being executed daily. This has made Tesla a "hard to borrow" security.

If however, a trader wanted to borrow shares of an "easy to borrow" stock like SPY ETF, which is a trust that seeks to provide investment results that correspond with the price and yield of the S&P 500, it will be much easier to obtain those shares. It's easy because of the large number of shares available for borrowing. The reason why there are so many shares available is because of the vast number of investors that are bullish on this ETF.

### 2.4 Current Communication

Currently, for SLAB traders to execute short sales they must create a proxy server. A proxy is a server (a computer system or an application) that acts as an intermediary for requests from clients seeking resources from other servers. SLAB traders must create this proxy because it is needed to connect with the Falcon software that SLAB uses. Falcon is able to receive requests and determine the number of shares available; it will be explained in detail in the technology tools section XX of this paper. The proxy is made off of Symphony and can be monitored using a security watch list. For the SLAB trader, when a locate request (which they are assigned) is submitted they receive alerts in Falcon (a pop up basically) informing them to act on it and send it back to the EQD traders. Each SLAB trader is assigned certain securities to monitor so that the team can get a response back as quickly as possible.

### 2.5 Advantages and Disadvantages

As our goal is to improve the communication process between EQD traders and SLAB traders we wanted to understand the strengths and weaknesses of the current process. We learned about how the current system works by meeting with equity traders and SLAB traders to understand what the users liked and disliked. The results of these meetings are summarized below. These meetings enabled us to incorporate the successful pieces of the current process while simplifying and eliminating any unnecessary work done by the traders.

### 2.5.1 Advantages

After understanding the current process we identified two major advantages:.(1) data flows between EQD and SLAB and (2) the process of approving and denying trades. We wanted our chatbot to keep these parts of the process to ensure we've created a better process that still implements the best parts of the old one.

#### 2.5.1.1 Data Flow

How the data flows between SLAB and the user is working well and doesn't need to be changed. User requests are quickly received and approved by the SLAB software and users are notified when a request has been approved and sent back. After meeting with both types of traders we learned that they felt the way they received messages was a problem but the way the data was being sent was not. Knowing this helped us understand that we didn't need to change the way the data is sent but rather the platform they are sending messages on.

#### 2.5.1.2 Acceptance and Declines

Because SLAB's approval process and response time is fast and efficient, the process for accepting and declining messages does not need to change. SLAB has its own process of approving which involves interaction with Falcon. We will not edit any part of how acceptance and declines are managed nor will we change the way SLAB is notified of incoming requests. Once a request is sent to SLAB it is their job to determine the amount of borrowing that is going to take place.

### 2.5.2 Disadvantages of Current System

Determining the key disadvantages of the current system was the most integral part of our project. Once we identified the major problems with the current system we were able to formulate our problem statement and begin planning the construction of our chatbot. The following sections summarize our findings.

#### 2.5.2.1 Not Completely Automated through Symphony

Symphony has become a major movement for BNP Paribas Global Markets. This evident in the $63 M USD investment that company has made.

"This investment positions us at the heart of an important development for the industry and signals our commitment to creating stronger and more efficient ways of working, using the latest advances in technology. The Symphony platform brings significant benefits, enhancing connectivity and enriching our ability to serve our customers more efficiently." –Cyril Cottu, Global Co-Head of Electronic Market Making and Commerce (Symphony 2018).

The Symphony platform, which is being rolled out globally to staff at the bank, is an innovative, secure and efficient communications and workflow automation tool with over 200,000 users across buy- and sell-side. Our understanding is that Symphony is the platform that will be the most used in the company once it is completely rolled out.

One of the elite features of Symphony is its ability to interact with chatbots. Specifically, a chatbot is an application that performs an automated task, such as setting an alarm, telling you the weather or searching online. Chatbots are used similarly to apps. Symphony chatbots are developed to work exclusively with Symphony and can be purchased in the market place, similar to the android or apple store.

By creating a chatbot on Symphony we will eliminate the need for any manual steps involved in the communication process. Users will no longer need to create a proxy, check for updates, or communicate directly with SLAB. The chatbot will do all the communication between the two parties.

#### 2.5.2.2 Process Speed

The second major disadvantage of the current system is response time. Not that the response time is slow, but rather it's not being done in the most efficient time possible. The creation and submission of requests done by SLAB and the time it takes between a trade being approved and the EQD trader being notified can be completely removed by the creation of a chatbot.

By removing the manual steps necessary to complete requests we will eliminate SLAB trader's communication and submission of EQD requests. Our chatbot will also send an immediate notification to the user removing time wasted between approval and notification.

# Chapter 3: Methodology

## 3.1 Problem Statement

After analyzing the current process, we identified two major issues within the current system. The first is that the process is too slow, and the second is the lack of integration into Symphony.

Currently, the submission process for requests is slow and outdated. The SLAB team shouldn't be completing tasks that can be done more simply by a chatbot. The manual input requests delay EQD traders' notification for request approvals. The EQD traders are not always notified exactly when their trade has been approved or declined which leads to slow downs when executing short sales.

We have been tasked by the Global Markets IT team to fix these two current process problems. One is the automation of the manual steps within the system. Global Markets would like the process to speed up and that will be done through automation and automatic responses. Secondly, Global Markets IT would like the entire process to go through Symphony. Integrating Symphony will put the process onto one system and will eliminate the extra steps taken to submit requests. It is also very important to BNP that Symphony is the communication tool that is used.

## 3.2 Objectives

To automate the trade process, our team has developed on major objective which can be broken down into two deliverables.

Objective: Develop a chatbot to automate the current communication process.

Correspondingly, the team had two deliverables which we discuss in detail in the next section.

### 3.2.1 Deliverable 1

This deliverable was about learning what the scope of the project we will be working on. Head of Global Markets IT Andrew Clark and his team gave us guidelines for creating a chatbot and walked us through how to connect to the software's we will be using. This deliverable was to learn how to automate the communication process described in Section 2.3.

### 3.2.2 Deliverable 2

There are 3 steps to complete this deliverable:

1: Create a Main function with language parsing that will serve as the front end communication for the user.

2: Create and define a list of methods that will interact with the main function and the API's to process requests.

3: Merge the list and main file, confirm it connects and works with both API's accordingly.

### 3.3 Technology Tools

In this section, we will discuss all the technology tools (programming language and IDE) we used, version control system and repositories, and information of all APIs we interacted with in this project.

#### 3.3.1 Programming Language and IDE

A programming language is a mix of commands, logic, and syntax in order to create a software program (Computer Hope, 2018). Programming languages includes "high-level language" and "low-level language". High-level languages are aimed to easy and understandable for a developer to code, and low-level languages are assembly and machine languages to allow a machine run successfully, but they are hard to read by human (Beal, 2018).

IDE stands for Integrated Development Environment. It is as a place to allow developers write and test software.

##### 3.3.1.1 Python

Python is a high-level programming language which includes a comprehensive library of modules and packages (McGrath, 2013). Python is the best programming language to be used in this project because it has a small learning curve. Its project structure is simply set up, and it has straightforward methods to post and get requests so that we could communicate with APIs in a convenient manner. The definition of API and how we are going interact with will be explained in section 3.3.4. We used Python 2.7 to write our main back-end source code which interacts with Symphony REST API and SOAP API. The source code includes a chatbot framework, a bridge that connects the chatbot and Symphony REST API, and workers work within the chatbot. Code is written in PyCharm IDE and programs are tested through Anaconda Prompt before being pushed into Git version control environment. All the environments will be explained in details in the following sections.

All third-party Python libraries we used in this project are accessed through Pypi Pip and are installed from BNP Artifactory library repo. Pip is a package manager to pull and install any library the program needs from a library repo webpage (W3School, 2018). The libraries we used are Logging, Suds, and Suds.wsse. Logging is a package to print messages on backend console while at the same time running code in PyCharm. It is useful to know what priority the message has, or where the message originates from. In this project specifically, it is used to log information of debugging issues. Suds allows the program to create a client with a valid Web Service Description Language (WSDL) path. Web Service Security (wsse) is the sub-library of suds that is able to create a soap-based object with a security token header and an XML message body.

##### 3.3.1.2 PyCharm

PyCharm is an IDE (Integrated Development Environment) specifically for Python programming published by JetBrains. PyCharm provides smart code completion, code

inspection, on-the-fly error highlighting, and quick-fixes, along with automated code refactoring and navigation functionalities (JetBrains, 2018). We used PyCharm 2018.2 Community Edition in our project.

### 3.3.2 Prompts and Commands

#### 3.3.2.1 Git Command

Git is a version control system that is used to manage any changes of files, codes, and programs. The operations in Git include push and pull from a repository, branching the project, and merging from several branches (Chacon, 2018). BNP uses Git instead of other version control systems because it is smaller and faster. Figure 1 shows the common Git commands and the visual relationships between Git commands and repositories.

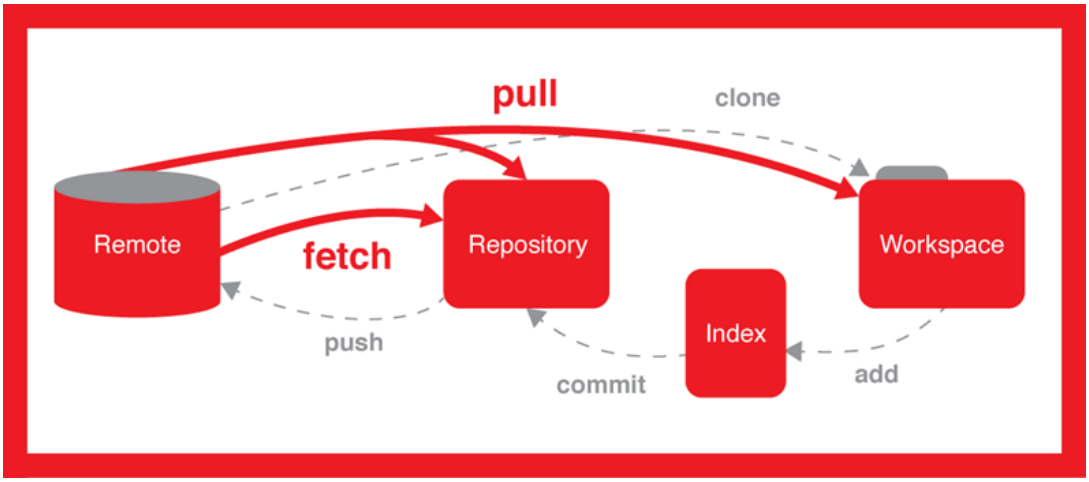


Figure 1 - Relationships between Git Commands and Repositories

#### 3.3.2.2 Conda and Anaconda Prompt

Conda is a powerful package manager and environment manager for several programming languages. To work with Windows operating system, Conda is opened through Anaconda Prompt, which is a command interface. It can also work with macOS or Linux by opening through a Terminal window. The Conda commands manage the installation of packages through Aconconda Prompt. It can query and search the Anaconda package status information, create new Conda environment, and install and update a package into existing environment (Continuum Analytics, Inc., 2017). The Conda command has several different syntaxes, but each syntax can only work on the corresponding operating system environment. In this project we used only Windows OS. This means Anaconda Prompt is the main interface we use to install Python Packages, test our Python 2.7 code, and run the final project.

### 3.3.3 Version Control Systems

Version control systems are software tools that help a software development team manage changes to source code. Version control keeps track of modifications of files, code, and access permission. The reason the function is called version control is because the system stores all updated versions of the project and when one version has an error, or needs to be disposed of, the user with specific administrative permission is able to restore the previous correct version without disturbing all other files and programs (Atlassian, 2018).

#### 3.3.3.1 BitBucket

BitBucket is a programming project repository we use. It is a version control platform for programming collaborations, and it is especially convenient to store private and confidential code. Our project code is stored under BitBucket BNP SymBot repository. To communicate with BitBucket, we use git commands in Anaconda Prompt (Bitbucket, 2018). Figure 2 shows the relationship and connections within a developer team, repository, and BitBucket server.

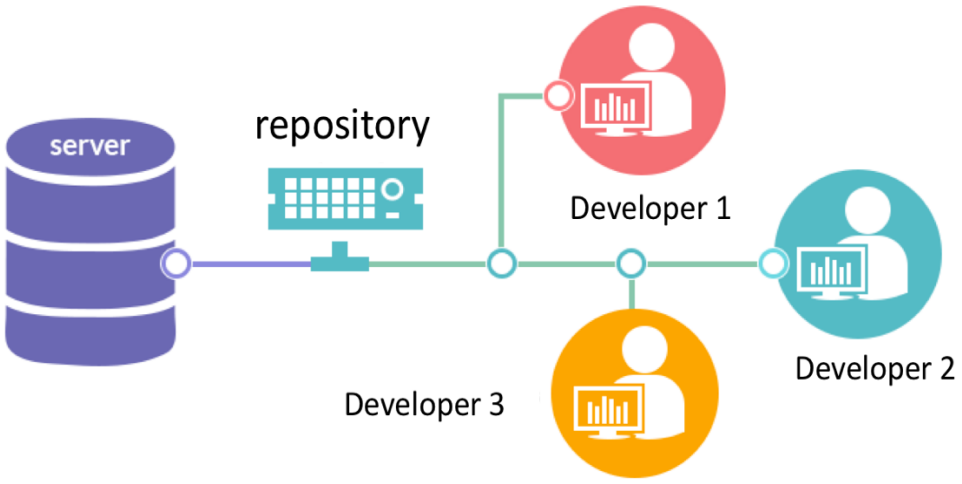


Figure 2 - Structure of Developers using BitBucket Server

#### 3.3.3.2 JFrog Artifactory

JFrog Artifactory is a repository manager which supports major packaging formats, build tools, and Continuous Integration (CI) servers. BNP chose Artifactory as their main repository manager because it supports all major packages. It is the only platform that can handle huge amounts of tools and version controls on an industry level. It allows automating development pipeline and deploying repositories in cloud or in a hybrid model. Figure 3 shows all the functions Artifactory manages as well as the relationship between them. In this project, we will be using Git and Bitbucket for Version Control and Python as our build tool (JFrog, 2018).

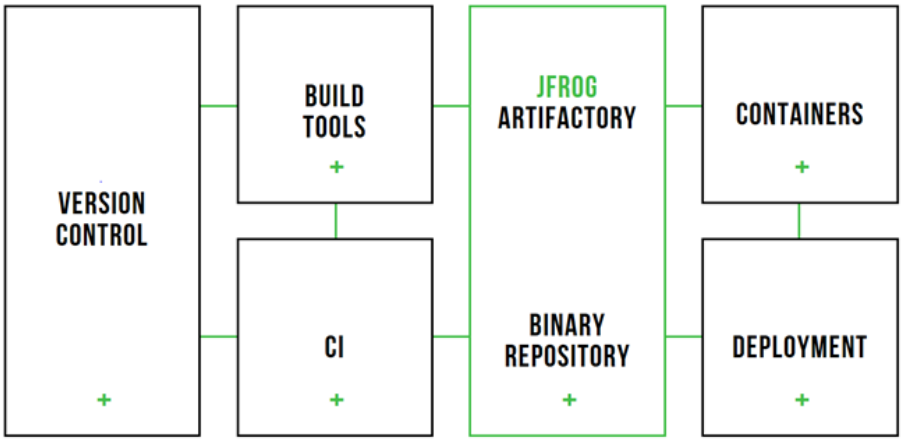


Figure 3 - Artifactory Functionalities

In this project, we used JFrog Artifactory to download and install BNP's internal Python packages as well as deploy the final product in production.

### 3.3.4 Application Programming Interface (API)

An Application Programming Interface (API) allows developers to get methods from an external software or website and utilize in their own application. API makes the use of eternal methods easier because developers do not need to know how the methods are created and how the logic inside of them looks like (Hoffman, 2018). We used several APIs to communicate with our chatbot in order to interact with third-party software and develop our chatbot within the software. Similar to the process described in the previous section, our chatbot communicates with Symphony software through Symphony API, and then through Falcon with Soap API in XML message. The Falcon software itself also interacts with Falcon API in order to achieve more BNP business purposes. The APIs we used will be discussed in the following sections.

#### 3.3.4.1 Symphony API

Symphony provides Extension API and RESTful API (Representational State Transfer). Extension API allows developers to build apps embedded within the Symphony's user interface, and the REST API gives developers access to all applications required to build a chatbot. We used REST API in this project to create a chatbot that can interact with Symphony interface and interpret the message input from Symphony chat room. REST API is implemented on several physical interfaces: the Pod is a cloud-based Symphony infrastructure, the Key Manager encrypts key messages from users, and the Agent encrypts and decrypts services for applications.

BNPP has many new functions and directories that would like to be added upon the original Symphony software, for example: directory of employees, explanation of acronyms, applications, and chatbots. To do so, each function or application is built through programs in back-end code, and all programs call the Symphony API to connect with the Symphony software. The goal of calling Symphony API is to implement all newly-added functionalities onto the user's side of the Symphony software. The software automatically executes the

functionalities with Symphony build-in front-end user interface without need to code and understand how to code it (Symphony, 2018).

#### 3.3.4.2 SOAP/Falcon API

SOAP API is a bridge between our chatbot and the Falcon software. SOAP API sends message using XML file format with columns storing the message type, attributes, and place to hold message content (Wodehouse, 2018). In our project, we will use SOAP to transfer data that includes client and request information. Figure 4 shows the structure of constructing a SOAP-XML object.

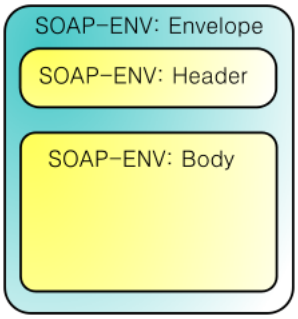


Figure 4 - Structure of SOAP-XML Object

# Chapter 4: Process Analysis

## 4.1 Architectural Overview

The simplified structure of the new process is demonstrated in the Figure 5. It shows the bridges that connect all environments and API's included within this project. The part our team was tasked to create was the chatbot. The chatbot is coded in Python and is stored under the Python Framework. The framework interacts with the Symphony messaging software through the REST API so it can retrieve any client input from the chatting interface. It then sends that input to the backend and generates a message to send back to the client. On the other side, our chatbot communicates with BNP's Falcon software, which is a stock trading and borrowing software used in BNP SLAB department. Falcon communicates through SOAP API by sending and receiving SOAP XML messages. By sending these messages it can interact with the methods inside the Falcon web service and determine the approval status of the user's request. Its methods include GetAvailability, GetApproval, and QueryApproval. After communicating with Falcon, our chatbot can send request approval messages through the chatbot and back to the client where request approvals can be accepted or rejected by the user.

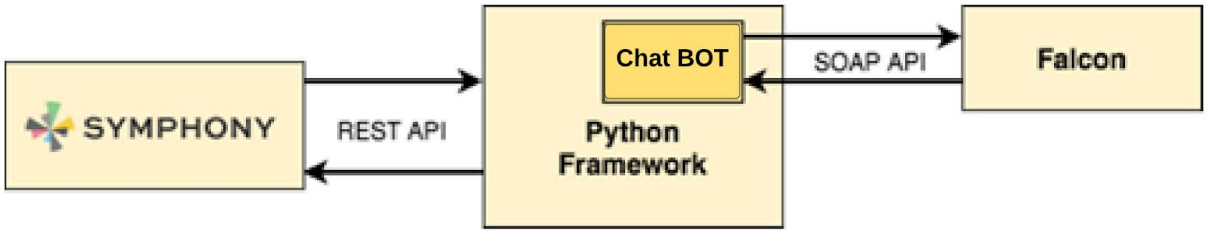


Figure 5 - Architecture of the new Process

### 4.1.1 Use Case Diagram

To get a simplified and graphical representation of what each entity does, we used the use-case diagram to provide a clear view of the system. A use-case diagram demonstrates the functionality of a system. It consists of actors and relationships. Actors are entities of the system. Lines in the diagram represent the interactions between different actors.

There are three actors in the chatbot project: client, chatbot, and SLAB Trader. A client sends a request and receives a response through the chatbot. The chatbot translates client requests and manages client data. Also, the chatbot translates the client's request into standard form. The chatbot can also send and receive messages from SLAB directly. SLAB traders make decisions based on clients' requests, and then generate the approval rate automatically (the approval rate could be generated together with SLAB's approval).

**Use Case Diagram**

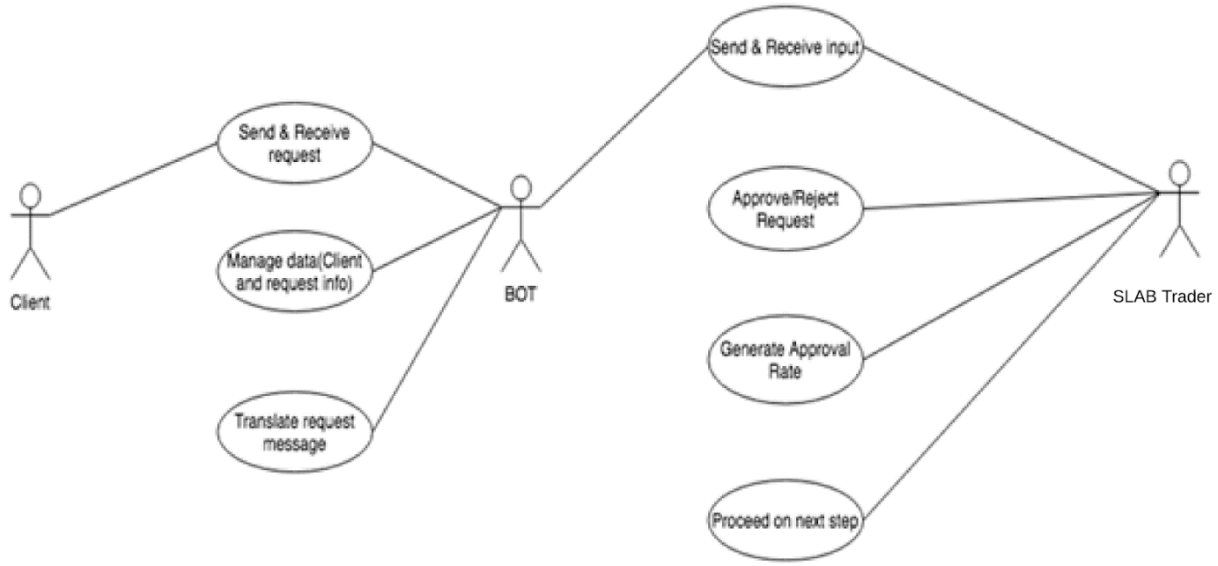


Figure 6 - Use Case Diagram

### 4.1.2 Swim-Lane Diagram

A swim lane diagram, sometimes called cross-functional diagram, indicates roles within a process (Masters, 2011). Each lane in the diagram is assigned to an actor or a phase in the process. The swim lane diagram is easy to understand especially for people not familiar with the process. The diagram is also easy to edit in order to identify redundancies. Because of these advantages, it is used in our project process design. Our group utilized the swim lane diagram to demonstrate the process of client communication through the Symphony chatbot. The process is categorized into four lanes: client, Symphony, chatbot and Falcon. Process starts from the blue circle in client lane and direction of flow follows red arrow. Process terminates at the pink circle. Therefore, the process starts from client sending request messages and ends up receiving approval message through Symphony.

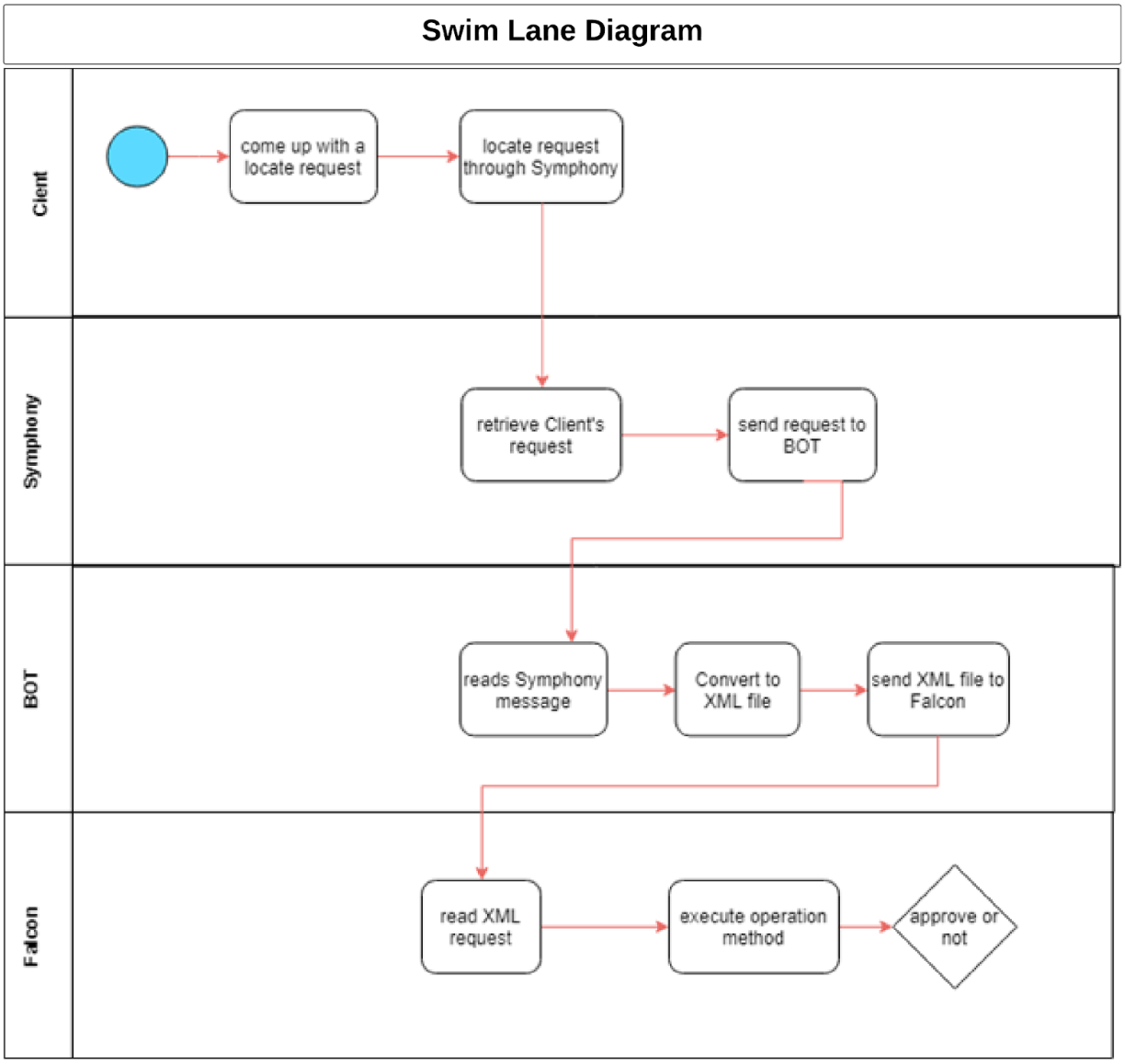


Figure 7 - Swim Lane Diagram -part 1

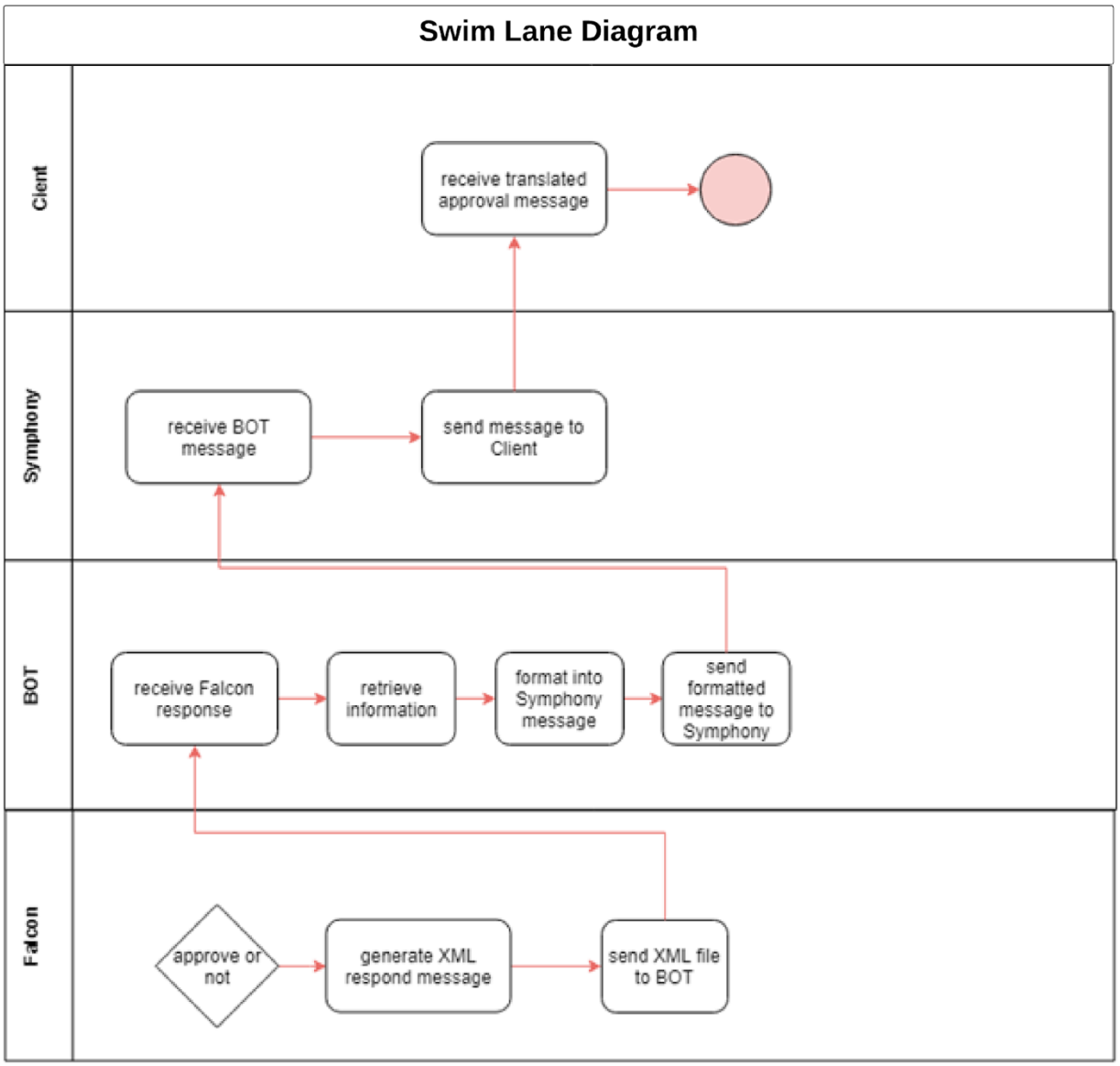


Figure 8 - Swim Lane Diagram - part 2

### 4.1.3 Data Flow Diagram

Data flow diagram is used in process designation of the new process. The blue rectangles represent entities, while gray rectangles represent processes. Texts above arrows are data flow following the arrow direction.

To achieve the data flow in this process, Symphony REST API and SOAP API are implemented. Rest API connects Symphony with the Framework, while SOAP API associates chatbot and Falcon. In this case, the chatbot can receive client's information and requests and SLAB can get the information accordingly.

The process could be seen into two directions: red arrow is the first direction, while the blue arrow represents the opposite direction. First, a client locates a request following the red

arrow, sending message through Symphony interface then delivered to Falcon and Falcon sending the translated request to SLAB traders. The second step follows the blue arrow. SLAB traders send back approval message to Falcon. Then Falcon sends back the message back to Clients through Symphony.

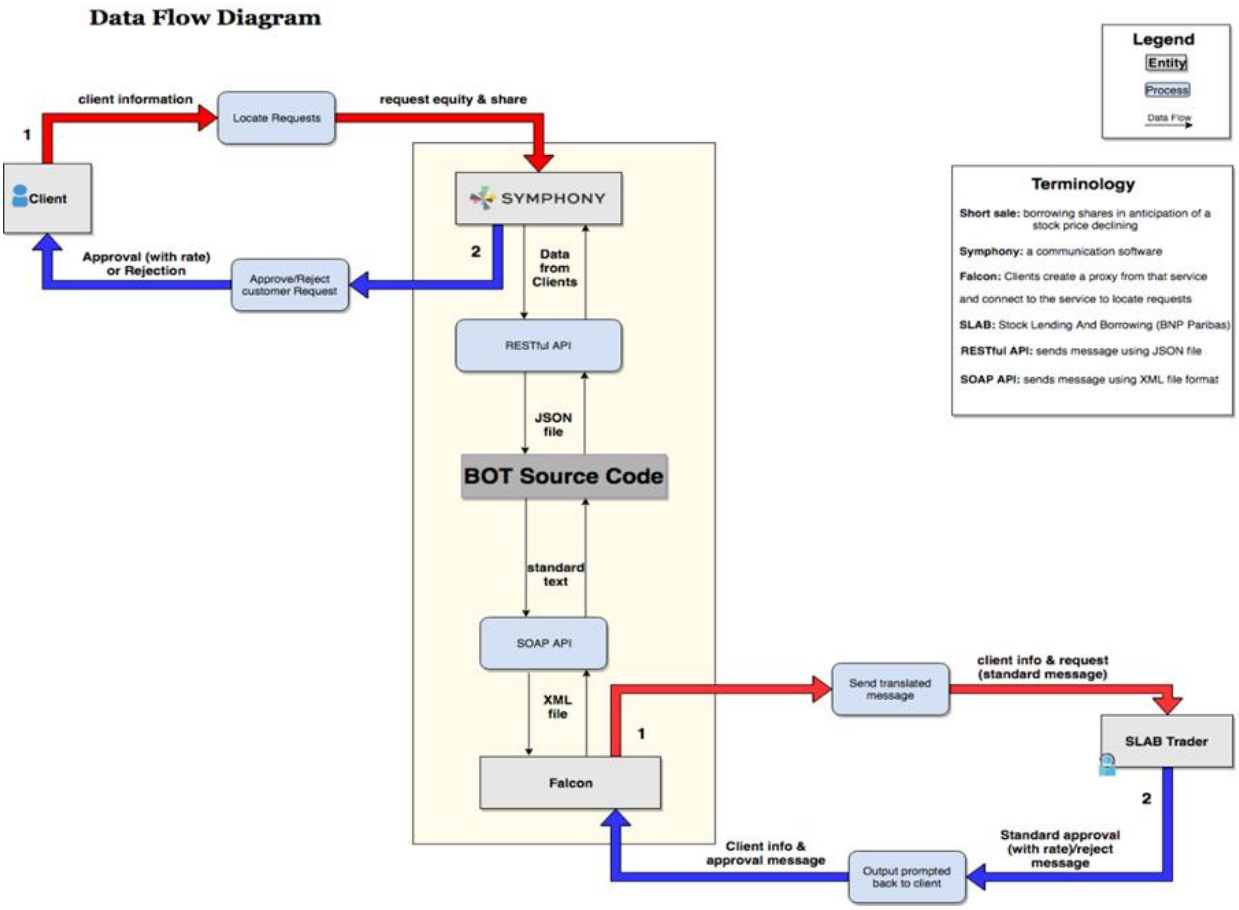


Figure 9 - Data Flow Diagram

## 4.2 Chatbot Process Design

SLAB belongs to BNP Paribas. Clients include both external clients (for example, investors in the market) and internal clients (for instance, Forward Desk, Volatility Desk, and Equity Desk). The word "locate" is used whenever a client requests to borrow shares from the sources the SLAB traders have access to. In the current process, a client must fill out a form and submit it to the SLAB desk; the SLAB desk then does its job and relays the answer to the trader through different software. The process is time-consuming, is hard to use, and doesn't work with the Symphony chat room. Our project aims to automate the locate request process for easy or hard-to-borrow equity. Rather than having clients upload their data through Short Sale Service client, we get rid of this process for clients and give them a more user-oriented Symphony prompted interface.

In our project, we are going to focus on the process of interaction between internal clients and SLAB Falcon through the Symphony Chatbot that we built. The way it will now work is as follows:

1. Clients contact the SLAB desk to see if they can borrow shares of a stock(s) they would like to short. A client sends a request message through Symphony. Symphony Chatbot receives both the client information and his/her request and translates the request(s) to standard input. Then the standard input is sent to SLAB Falcon. Falcon decides whether they can approve or decline client's requests.
2. If yes, an approval message including approval rate is generated automatically.
3. If yes, but only a partial number of shares has been granted, an approval notice will be sent to the user as well as the partial amount of shares they can borrow and approval rate.
4. Otherwise, the request is turned down and a rejection message is sent back to the user. Falcon prompts back all output messages to the client through Symphony Chatbot.
5. The client should receive a message showing SLAB's approval or rejection. Then SLAB can proceed the transaction.

### 4.2.1 Flowchart

The flowchart is a formalized representation of a logic sequence (Rouse, 2016). In the process design, we utilized flowchart to demonstrate the detailed structure of the request sequence. The sequence begins and stops at the green ovals, which are terminators of the flowchart. Then the sequence follows the arrow directions. Blue rectangles are processes and decisions are manifested in diamond shapes. A client opens Symphony and starts a conversation with the chatbot. The chatbot has two categories of functionalities: request transaction and query service. The left part of the flowchart represents one functionality: locating request. A client enters a request message. For instance, "GOOG,100". Availability is the first thing to check. If the requested equity is not available, there will be "Not Available" response. After that, a timer will record user waiting time. And then check if the request could be approved. Before the chatbot responds time out, check if the request message is approved. If it's approved, proceeds transactions. On the right side, there three different processes: QueryMostActiveSec, QeuryApproval, and QueryApprovalSysNameAndTimeStamp. QueryApporval has four sub-processes after it. Finally, query results will be sent back to the client.

## Flowchart

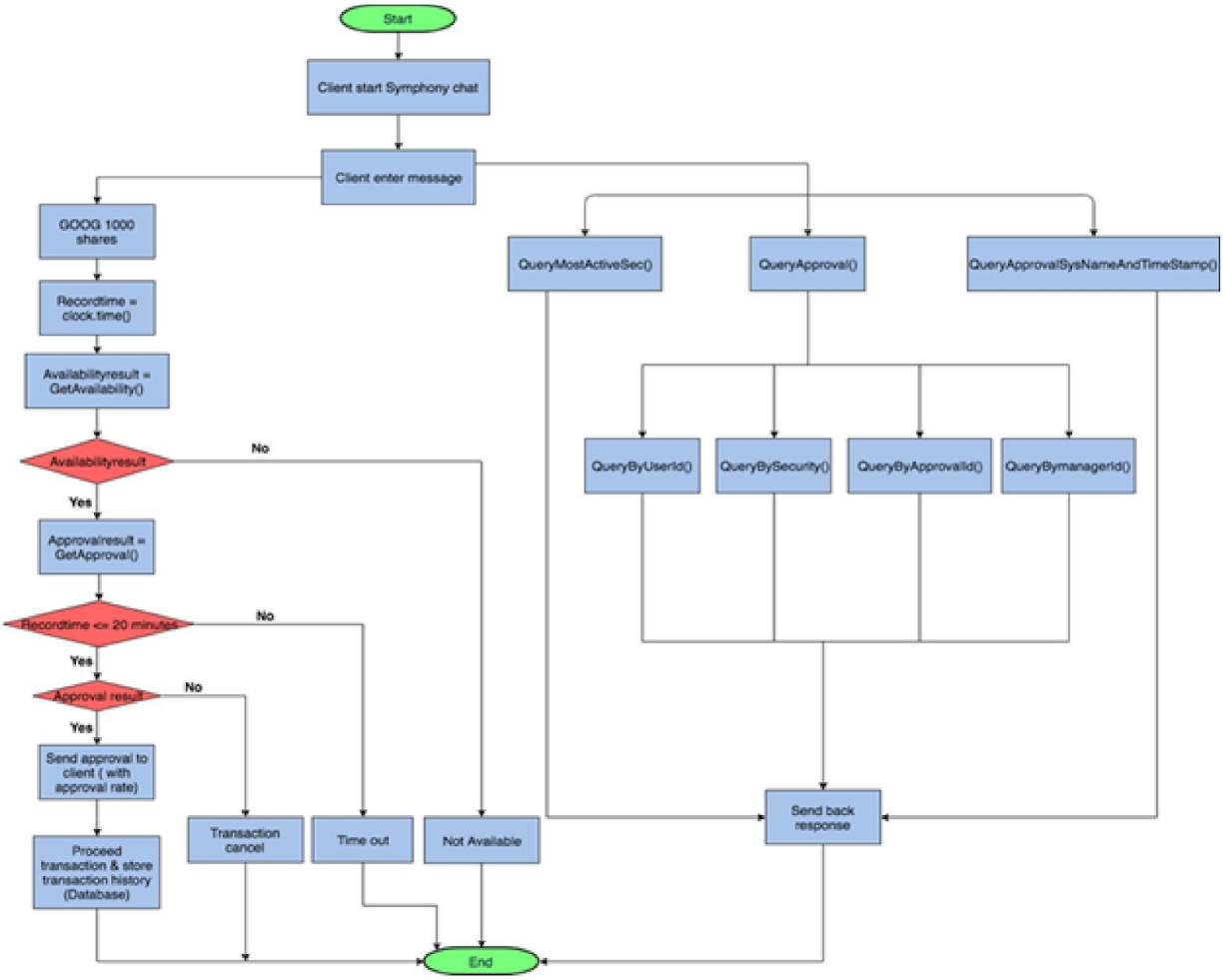


Figure 10 - Flowchart of Calling Falcon Methods

# Chapter 5: System Analysis

## 5.1 Message Processing

When a user types in a text message in Symphony, the text is received in Symphony chat room communicated by Symphony API. The framework developed by BNP IT team is able to convert a Symphony HTML format message into a string type object that is able to be used by the chatbot. To interpret the message string, the chatbot first calls function split( ) to separate the string into words, then it stores the words into an array list known as 'wordList', and distinguishes the array list wordList by cases.

The chatbot is also able to recognize several commands: 'Hi', 'Method', 'help', 'bye', 'stock request', 'status', 'search', and any input that is not programmed by the chatbot. We will discuss about how each command work in the following subsections.

### 5.1.1 Language Processing

All of the commands in this section are aimed to allow the chatbot to interact like a human to give users a better experience. When the user input matches the following cases, the according command is triggered in highest priority.

#### 5.1.1.1 Hi

This command can be triggered if the input equals any of the following cases: "Hello, hello, HI, Hi, hi, Hey, hey, Yo, yo, Bonjour, and bonjour." The respond of command 'Hi' is to give a brief self-introduction of the chatbot and introduce a few commands that are commonly sent by the user. This is helpful since it gives new users the ability to understand what the role of the chatbot is and how it is helpful. Figure 11 shows the response message generated by the chatbot when the user enters 'Hi'.

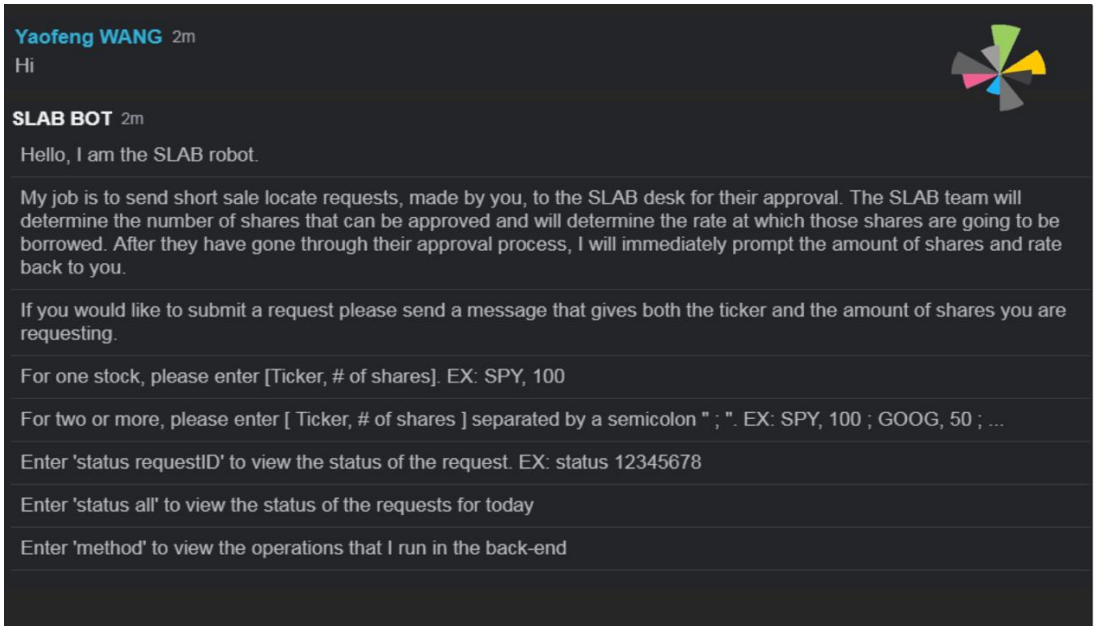


Figure 11 - Interface of command "Hi"

#### 5.1.1.2 Method

This command can be triggered if the input equals any of the followings cases: ‘Method’, ‘Methods’, ‘method’, and ‘methods’. The chatbot responds to the command ‘Method’ by showing the user the Falcon methods used in the chatbot. The falcon methods are methods listed in Falcon web service (Falcon API). This command is used specifically as a development tool for development team at BNP. Figure 12 shows the response message generated by the chatbot when the user enters ‘method’.

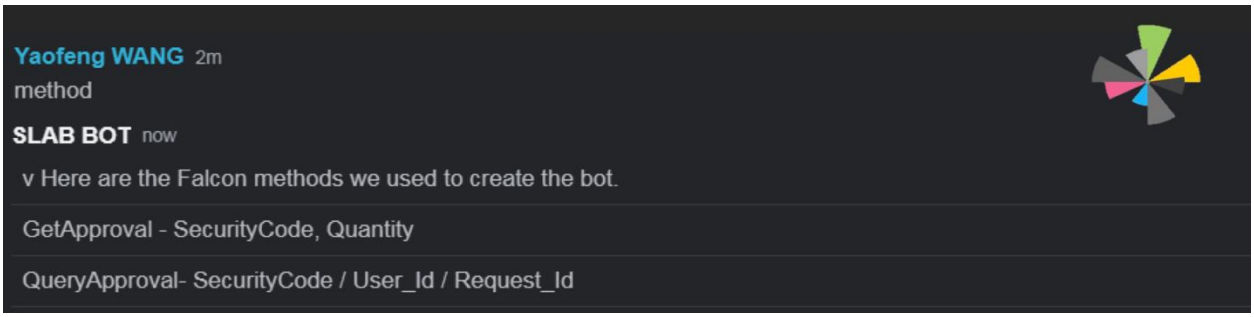


Figure 12 - Interface of Command "method"

#### 5.1.1.3 Help

This command can be triggered if the input equals any of the following cases: ‘Help’, ‘help’, ‘SOS’, ‘Sos’, and ‘sos’. The command ‘Help’ gives the user a full list of commands, the alternative ways to type them in, and the direction of each command. This is the official list to refer to when a user needs help.

#### 5.1.1.4 Bye

This command can be triggered if the input equals any of the following cases: ‘Bye’, ‘bye’, ‘bye bye’, ‘Goodbye’, and ‘good bye’. The aim of this command is to make the chatbot more ‘human-like’. The chatbot will respond back with a message that say bye back. Figure 13 shows the response message generated by the chatbot when the user enters ‘bye’.


Figure 13 - Interface of Command "bye"

### 5.1.2 Using the Methods to Short a Stock

In order to short a stock, a user can type in commands in two different ways: a single request and a list of requests.

For a single command, a user should type in a request command that is structured by "stock name" (string), "comma", and "number of shares" (integer). For example, if a user wants to borrow 100 shares of Google, he or she should enter 'GOOG, 100'.

Commands in a list are separated by semicolon. For example, if a user wants to short 100 shares of Google, 101 shares of IBM, and 102 shares of Tesla, he or she should type in "GOOG, 100; IBM, 101; TSLA, 102" or "GOOG, 100; IBM, 101; TSLA, 102;". If there are any invalid commands in a list, for example: "GOOG, 100; xxxxx; IBM, 101; ;;1111111; TSLA, 102", the result is still the same because any invalid commands in a list are ignored without processing the other commands in the list. More error handling cases will be discussed in section 5.2.

The chatbot handles the stock request cases by looking at the input. If the input is able to be split by ',' and the number of elements equals to two, it is a single request. To form a valid request, the first element should be a string, and the second element should be a positive integer less than 99999999 to avoid exceeding the limit amount from Falcon. Any format other than this in this case will not be processed.

When a request is received in the chatbot, the chatbot constructs a SOAP XML message that contains all columns of data needed and the method getApproval( ) from Falcon. It then sends these messages to Falcon with an identity ID that is represented by the user ID of the Symphony user. By containing an identity ID in each request, the chatbot is able to know the sender of a request easily by tracking the ID to do any calculation, error notice, or statistics. If the request is processed as an easy-to-borrow case, the response message is returned immediately in a SOAP XML format. The chatbot will then take out a few key pieces of information from the input message and then interpret those inputs. These inputs are the: request ID, stock name requested, number of shares requested, number of shares approved, approval status, and rate approved. The information is stored and is constructed into a HTML message to be printed back to user. The approval statuses are differentiated in colors: "APPROVED" is shown in green, "PARTIAL APPROVED" is shown in yellow, "DECLINE" is shown in red, and "PENDING" is shown in orange.

If the approval status sent from Falcon is 'Waiting", it is a hard-to-borrow case. In this case, the status can't be changed unless a SLAB trader manually approves or declines it on the side of Falcon. In this case, instead of directly returning the message back to the user with a waiting status, the chatbot prompts back a message by calling send() method in framework. The message says: "This is a hard-to-borrow request. I am working on it. I will let you know when the information is ready."

After this message is sent, the chatbot will track the status of the this request by calling Falcon method QueryApprovalByApprovalID() using the corresponding requestID as the parameter. The chatbot calls QueryApprovalByApprovalID() every 10 seconds to check if the status has changed and will stop after 20 minutes is passed. If the status is changed, the HTML

respond is formatted right at the time and is send back to user. If the status is not changed after time is up, the message “Time is up! Please contact Slab desk at (XXX) XXX- XXXX!” is sent back to user.

When the user sees this message, he or she can call the slab desk to notify the traders to manually approve the request for them. Both the time interval and the time limit are global variables so that the time can be improved to be more appropriately after the chatbot works for a few months in production.

If there is ‘;’ in the input, it is a list of requests. All the requests are stored in a list to be processed one by one by a for loop. Each request is examined following the same procedure in the last paragraph to confirm that it is a valid request. If there is no valid request in a list, the chatbot will return back with a message that says “Oops, I don’t understand.” If there is a valid request in a list, all invalid requests in the list are ignored, and only the response of the valid requests are returned.

If there are one or more hard-to-borrow cases in a list of requests, each hard-to-borrow request is processed in a thread created for this specific one. Threading was incorporated into our program because it allows all hard-to-borrow cases to work at the same time without disturbing the process of the easy-to-borrow requests.

The whole process is described by the flow chart in Figure 9.

### 5.1.3 Status

‘Status’ command is used when a user wants to query a request information. There are two ways to use ‘status’: ‘status RequestId’ and ‘status all’. The details of these two commands will be discussed in the following sections.

#### 5.1.3.1 Status RequestID

Each request is submitted with a request ID recorded into the Falcon system. The command of “status requestID” allows users to access the information of any request submitted as long as the user has the corresponding request ID. For example, if a user requested 100 shares of Google, after he sends “GOOG, 100”, a requestID that equal to “12345678” with the request status are provided by the chatbot. If the user wants to look at the request status in the next few days, he or she may enter “status 12345678” to receive a response with a one-line record of this request.

When executing the command, the chatbot calls QueryApprovalByApprovalID() method to Falcon with the requestID as the parameter. Falcon responds with the information of the request including request status, request stock, rate approved, number approved, and number requested. The chatbot takes in the data and constructs a HTML table that lists the data and prints it back to user.

Currently, a user can access any request record by using the requestID without having a credential security control. This means a user is able to see the request sent by the other users.

After some testing in production BNP will decide whether the credentials are needed for specific users.

#### 5.1.3.2 Status All

To access all requests made that day by user he or she can type "status all" to see a full list of records. This command is helpful for a user to track the requests they issued by the day to compare the rate, number of shares, approval status, and make a calculation of all requests. A user is also able to keep track of any update of the hard-to-borrow request. The full list of request status is constructed by using HTML table so that the structure is simple and straightforward. The table includes a header of titles, index of each record ordered by time, and different colors of request status differentiated by APPROVED, PARTIAL APPROVED, and DECLINED.

This command is achieved by calling queryApprovalByUserId() with the userID of the sender as the parameter to call the function. The method returns the most recent records in 24 hours. The chatbot filters the records to keep only the records issued by the calendar date of the day and construct the HTML table to respond to user. If there is no request sent in the day, a message is sent to user telling him that he did not send a request.

### 5.1.4 "I do not Understand"

If the input is not matched by any one of the cases discussed above, or if the input is not a valid request or status search, the chatbot returns back with a message of "Oops, I don't understand!" with a way to correct the input if there is any. More details will be discussed in section 5.2.

## 5.2 Error Handling

### 5.2.1 Invalid Single Request

The format of a single request is "stock name (String), number of shares (integer)". If the string user entered can be split by coma ',' and the number of elements after split is equal to 2, the string is considered in the case of single request. There are several situations that violate this case and cause an error. The type of error message to prompt back is displayed in table 1.

| Command typed by user | Respond send from bot |
|---|---|
| GOOG, 100 (Right format) | |
| GOOG, GOOG | The second arg should be a positive integer! |
| 100, GOOG | The first arg should be stock name! |
| 1,1 | Sorry I don't understand! |
| GOOG, 999999999999999 | The number requested exceeds the max limit! |
| GOOG, -2 | The second arg should be a positive integer! |
| GOOG, 100.3 | The second arg should be a positive integer! |

Table 1 - Output of Invalid Single Requests

### 5.2.2 Invalid Multiple Request

The format of a multiple request is "stock name (String), number of shares (integer); stock name (String), number of shares (integer); ….". If the string contains semicolon ';', the string is considered in the case of multiple request. There are several situations to violate this case and cause an error. The type of error message to prompt back is displayed in table 2.

| Command typed by user | Respond send from bot |
|---|---|
| GOOG, 100; IBM, 101; TSLA, 102 (Right format) | |
| GOOG, 100; IBM, 101; TSLA, 102; (Right format) | |
| ; / ;; / ;;; / … | Sorry I don't understand! |
| 1;1 | Sorry I don't understand! |
| GOOG, 100; IBM, 101; TSLA, 102;;;;; | Respond same as the right format |
| GOOG, 100; IBM, 101; TSLA, 102; 111 | Respond same as the right format |
| GOOG;100 | Sorry I don't understand! |

Table 2 - Output of Invalid Multiple Requests

### 5.2.3 Invalid Status Request

The format of a status request is "status requestID (integer)" or "status all". After the string is split by blank space ' ', if the first element is "status", the string is considered in the case of status request. There are several situations to violate this case and cause error. This type of error message to prompt back is displayed in table 3.

| Command typed by user | Respond send from bot |
|---|---|
| status 12345678 (right format) | |
| status all/ status ALL / status All (right format) | |
| status | Missing the second argument |
| status GOOG | The second arg is not an integer! You may enter "status all" |
| status 123 | The requestID is not valid. Unable to query 123. |
| 12345678 status | Sorry I don't understand! |

Table 3 - Output of Invalid Status Request

### 5.2.4 Invalid Commands

If the command entered by user does not fit in the case of request, status, hi, bye, help, or method, the chatbot is unable to recognize the command, and the command is treated as invalid command. The only response for invalid commands is "Oops I don't understand you"

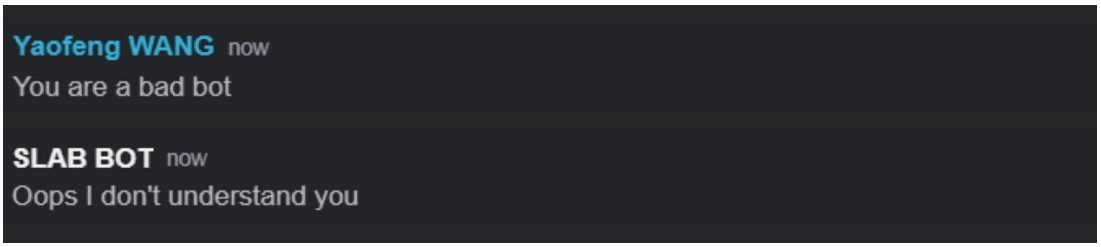


Figure 14 - Output of an Invalid Command

### 5.2.5 Server Error

When the server of Falcon is broken, or when Falcon is in a testing environment, both request and status cannot be used. In this case, a message will prompt user that “Falcon is down! Please try later or contact the slab desk!” When Falcon is back, all functionalities will resume to work normally.

# Chapter 6: Results

## 6.1 Build a Functional Chatbot

In the seven weeks we worked on this project we had one main goal: Construct a chatbot that BNP could implement into its working environment. We feel in the brief time we had that were able to complete this task. Most importantly we have created a chatbot that works. Not only does it work but also its functionalities seem to improve the speed at which traders conduct short sales. Although we don't have concrete testing completed our sponsors believe that what we have created is much faster and easier to use.

## 6.2 User Friendly Interface

We have created a chatbot that is very simple to use by focusing the majority of our time constructing a strong natural language processing component. The natural language processing allows for multiple inputs and simply asks users to try again if they manage to not input messages into the chatbot correctly. Our chatbot gives simple yet specific instructions on what to type in and how to get what you're looking for. Most importantly the chatbot is easy to use when conducting short sales. So much so, users can execute short sales in under 10 seconds on certain easy to borrow requests. On top of that its friendly in its own way. If you send the message "bye" to our but it will respond "bye, bye". If a user says "Bonjour" it will prompt the "Hello" message shown earlier.

## 6.3 Special Features

We also created three special features that were not requirements for our project. We created them because we felt it made the chatbot more likely to get adopted among the workers at BNP. By adding features that would make the chatbot easier and more practical we feel the special features were a significant addition to the final project.

1. Status: Asking the chatbot to find one specific short sale request using and ID number.
2. Status ALL: Asking the chatbot to find all the requests made that day by that specific user.
3. Inputting multiple trades: Asking the chatbot to perform a short sale locate request for multiple stocks that are separated by a ';'.

# Chapter 7: Recommendations

## 7.1 Testing

According to the testing conducted by IT consultants, the Symphony chatbot passed all technical test cases. This means the chatbot currently is technically feasible. Since our time was limited, we were unable to conduct the proper amount of user testing on our chatbot. More testing must be done on the chatbot to ensure that it is easy to use and can indeed speed up communication for Users and SLAB Traders. For example, we recommend letting a team of SLAB traders use the chatbot and persuade clients to choose Symphony chatbots as their primary messaging interface. We hope this offers some advantages for sample users. Furthermore, we should ask for feedback from SLAB traders and clients separately. The feedback we received should include the duration of wait time during one single request, willingness to use this chatbot in the future, and any difficulties while using it.

## 7.2 Falcon

We also recommend adding a cancel feature into the bot. Currently, we focused on GetApproval(), QueryApproval() and GetAvailability() methods, which are within the 14 methods of Falcon. Falcon does not currently include a cancel method. When considering the circumstance when clients type the wrong equity or number of shares, or doesn't want to wait anymore he or should be able to cancel. If the cancel method is added to Falcon, clients are able to terminate the request while waiting for the response.

## 7.3 Chatbot

We have developed and utilized two Falcon methods GetApproval() and QueryApproval(), other twelve built-in functions should be added into the chatbot. Furthermore, it would be good to explore SLAB functionalities. For instance, the chatbot could provide approval rate comparisons within the last three transaction days and classification service, which is supported by the database at the backend. Lastly, natural language processing needs to be developed further. By adding more language processing options into the framework, clients will be able to have a better user experience by decreasing the number of errors during conversations.

# Chapter 8: Conclusion

The Global Markets IT division of BNP Paribas currently conducts short sale locate requests that are complex, time-consuming, and labor intensive. The process lacks a single system to input allocations, an easy way to audit the activity, and a way to track multiple versions of the data sources; thus, our group designed a chatbot that would improve this process by addressing bottlenecks to reduce the amount of time required from the equity and derivative traders and the stock lending and borrowing team.

The new process consists of a web-based front-end application - which is for users to input their allocation keys and for the ITO Business Management team and management to view allocations and history. The new design addresses the problems that were identified by management and the ITO Business Management team; it provides a chatbot that improves the overall process by speeding up the interactions when executing short sales. In addition to the designs and database, significant documentation and system analysis were provided for future reference and for future extensions of this project. Yaofeng Wang, the lead developer for this project, is also a double major in industrial engineering and will continue to look for new ways to improve the chatbot we created.

# Chapter 9: Reflection on the project

## 9.1 Discussion of Design

In the project, we designed a detailed process. First, we utilized use case diagram to evaluate components' functionalities. After identifying the basic tasks of the project, the swim lane diagram helped me to understand the process and map out every single step in the process. The swim lane diagram categorizes subprocesses in four lanes: Client, Symphony, Chatbot, and Falcon; while the diagram also has disadvantages in process design: there are some unnecessary steps demonstrated. Thus, data flow was chosen to demonstrate all processes needed and sufficient information, especially data flow can be found above arrows. What's more, the data flow diagram includes interaction between entities: APIs perform as a bridge to connect the Symphony platform, chatbot, and Falcon. After we designed the architecture of the process, we moved to the technical part of the design. To realize chatbot functionalities, a detailed logic plan is crucial. We utilized flowchart about the logic of chatbot, addressing the relationship between Falcon built-in methods. We added a timer in the process, which checks the waiting time twice and determines "time-out" condition to make sure user wait time won't be too long to get a concrete response. Basically, we first analyzed the new process and then designed the chatbot architecture using a flowchart.

## 9.2 Discussion of constraints considered in the design

During the project designation, implementation before the project is a constraint that must be considered. BNP Paribas has a strict security policy to protect the information of the company. The duration of our group in the company is also limited. There is not enough time for our group to request Falcon implementation on our desktop. Our group can test easy-to-borrow equities on the testing environment, while for the hard-to-borrow equity tests we cooperated with some SLAB traders. In the tests, we requested hard-to-borrow stocks in the testing environment, and had SLAB traders to approve; thus, the tests were handled properly.

The second constraint is that we have to consider the client's waiting time. In order to appeal clients to locate request, the chatbot response has to be fast. To overcome this constraint, we added a timer in the chatbot. If the waiting time exceeds 20 minutes, the chatbot prompts back client a message, which asks client to call SLAB desk directly. After the implementation of the timer, another problem arises: If a client has a list of equities to request, especially including hard-to-borrow stocks, it's too time-wasting to wait for every request terminated. Therefore, multithreading becomes an efficient alternative. Multithreading enables several requests to run at the same time. With the design of timer and multithreading implementation, users can locate a list of requests in succession and don't need to wait for a long time.

## 9.3 Discussion of the need for life-long learning

The overall experience of the project is both challenging and rewarding. We learned to analyze the real-world process on different levels. When we first got in touch with the project, we were not familiar with the finance industry and related software. To evaluate the current

process and identify our project goals, we learned to use Symphony, understand locate requests, SLAB desk functionality. We utilized use case diagram to help me understand basic process components; then we mapped out the swim lane diagram and data flow diagram for architectural design. Although we were equipped with some knowledge of the diagrams, process analysis on different levels, from component functionality to architectural design, is fresh new for me.

What's more, we learned how to adjust the project to user experience, technical feasibility, and other occurring problems. Through conversations with SLAB traders, IT colleagues, and supervisors, we deducted process component which is already existed in the company, changed the process design for technical reasons.

The most important thing we learned is how to cooperate with colleagues. We worked with teammates who have different academic backgrounds. How to combine our advantages and how to assign work are crucial for the project. Also, we received a lot of help from other colleagues, supervisor, and professors. Conversations with all of them provided me sparkling light in work.

From this project experience, we learned to analyze a real-industry project by using industrial engineering knowledge, we learned to adjust our design of the process according to all sorts of conditions, and most importantly cooperated with others. We will continue to combine knowledge and practice, and welcome new challenges.

# Bibliography


- Atlassian (2018). What is version control. Retrieved from https://www.atlassian.com/git/tutorials/what-is-version-control.
- Bartholomew, H. (2018). Symphony bots march on Bloomberg. *Risk*. Retrieved from https://symphony.com/documents/risknet-symphony-bots-march-on-bloomberg.pdf.
- Beal, V. (2018). High-Level Language. Retrived from https://www.webopedia.com/TERM/H/high_level_language.html.
- Beers, B. (2018). Short selling basics. Retrieved from https://www.investopedia.com/articles/investing/100913/basics-short-selling.asp
- BitBucket (2018). Git, Your Way. Retrieved from https://bitbucket.org/product/features.
- Chacon, S., Long, J. (2018). Git Local-Branching-On-The-Cheap. Retrieved from https://git-scm.com/.
- Computer Hope (2018). Programming Language. Retrieved from https://www.computerhope.com/jargon/p/proglang.htm.
- Continuum Analytics, Inc. (2017). Conda. Retrieved from https://conda.io/docs/.
- History of the bank over the last two centuries. (n.d.). Retrieved from http://group.bnpparibas/en/group/history-centuries-banking.
- History of the Group. (n.d.). Retrieved from http://www.bnpparibas.com.sg/en/bnp-paribas/bnp-paribas-group/history-of-the-group/.
- Hoffman, C. (2018). *What is an API.* Retrieved from https://www.howtogeek.com/343877/what-is-an-api/.
- McGrath, M. (2013). *Python in easy steps.* In Easy Steps.
- JetBrains (2018). PyCharm Getting Started. Retrieved from https://www.jetbrains.com/pycharm/documentation/.
- JFrog (2018). JFrog Artifactory Enterprise Universal Repository Manager. Retrieved from https://jfrog.com/artifactory/.
- Kilpatrick, K. (2018). bot
- Innovate Asia 2018 Hackathon. Symphony. Retrieved from https://symphony.com/blog/item/symphony-innovate-asia-2018-hackathon.
- Masters, M. (2011). An Introduction Swimlane Diagrams. *Modern Analyst.* Retrieved from https://www.modernanalyst.com/Resources/Articles/tabid/115/ID/1868/An-Introduction-to-Swimlane-Diagrams.aspx.
- Norton, S. (2017). BlackRock Using Symphony Communications Platform to Chat With Other Firms. The Wall Street Journal. Retrieved from https://blogs.wsj.com/cio/2017/04/06/blackrock-using-symphony-communications-platform-to-chat-with-other-firms/.
- Quan, W. (June 2018). J.P. Morgan: Enabling Content Search UX Through the J.P. Morgan Markets Search Bot. Retrieved from https://www.youtube.com/watch?v=DSSpPWQJLmU.

- Retail Banking & Services. (n.d.). Retrieved from http://www.bnpparibas.com.co/en/about-the-group/activities/retail-banking-services/.
- Revolvy, L. (n.d.). "Symphony Communication" on Revolvy.com. Retrieved from https://www.revolvy.com/page/Symphony-Communication.
- Rouse, M. (2016). Flowchart. *Tech Target*. Retrieved from https://whatis.techtarget.com/definition/flowchart.
- Symphony Blog. (n.d.). Retrieved from https://symphony.com/blog/item/introducing-symphony-partners-dow-jones-mcgraw-hill-financial-and-selerity.
- Symphony (2018). Overview. Retrieved from https://rest-api.symphony.com/docs/rest-api-introduction.
- W3School. Python PIP. Retrieved from https://www.w3 schools.com/python/python_pip.asp.
- WodeHouse, C. (2018). SOAP vs. REST: A Look at Two Different API Styles. Retrieved from https://www.upwork.com/hiring/development/soap-vs-rest-comparing-two-apis/.