CurrencyHFT

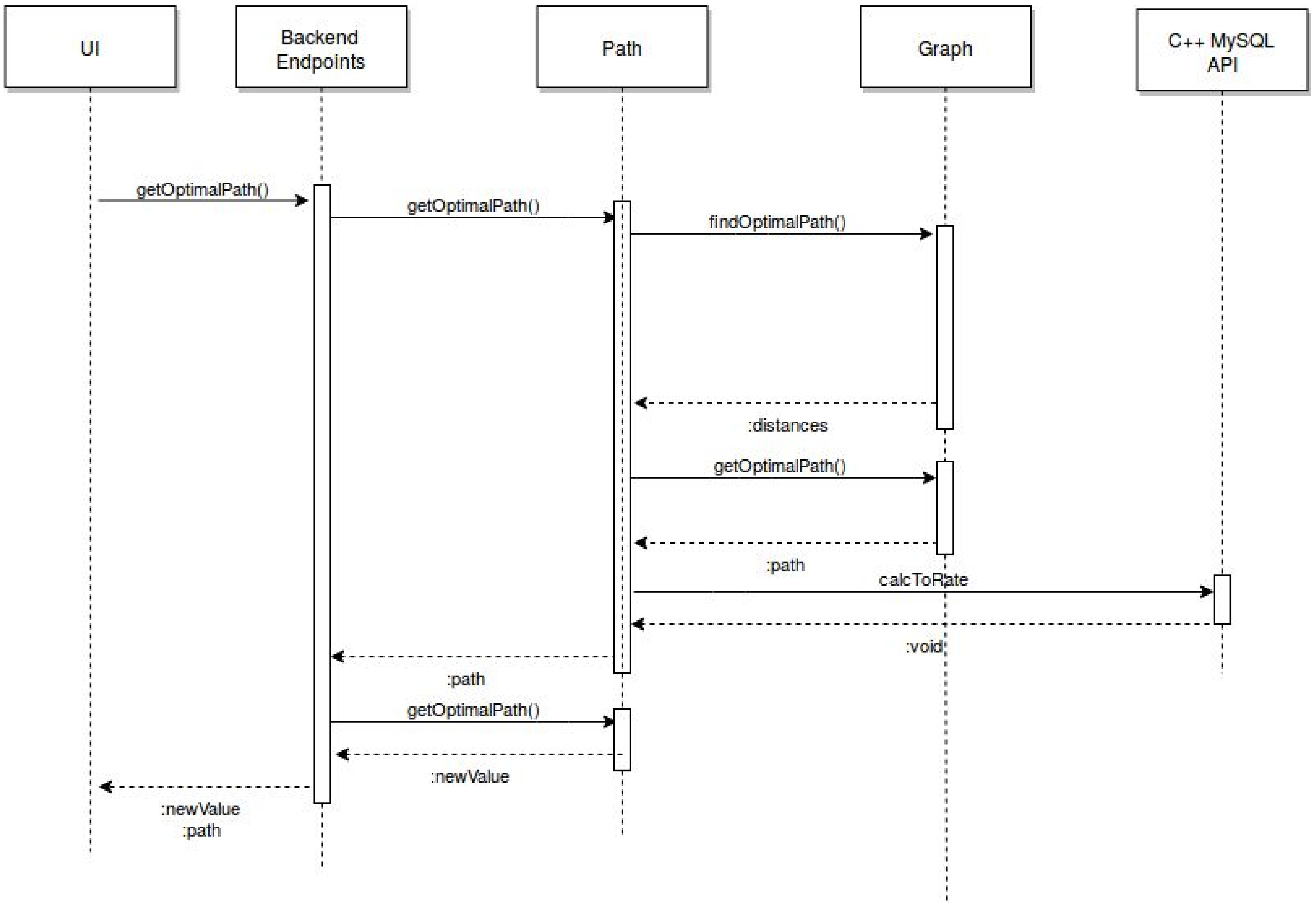
Cha Ching

STAKEHOLDER REVIEW 1

Jack Cusick
James Nakashian
Patrick Engelsman
Matthew Cordone
Ayushi Mishra

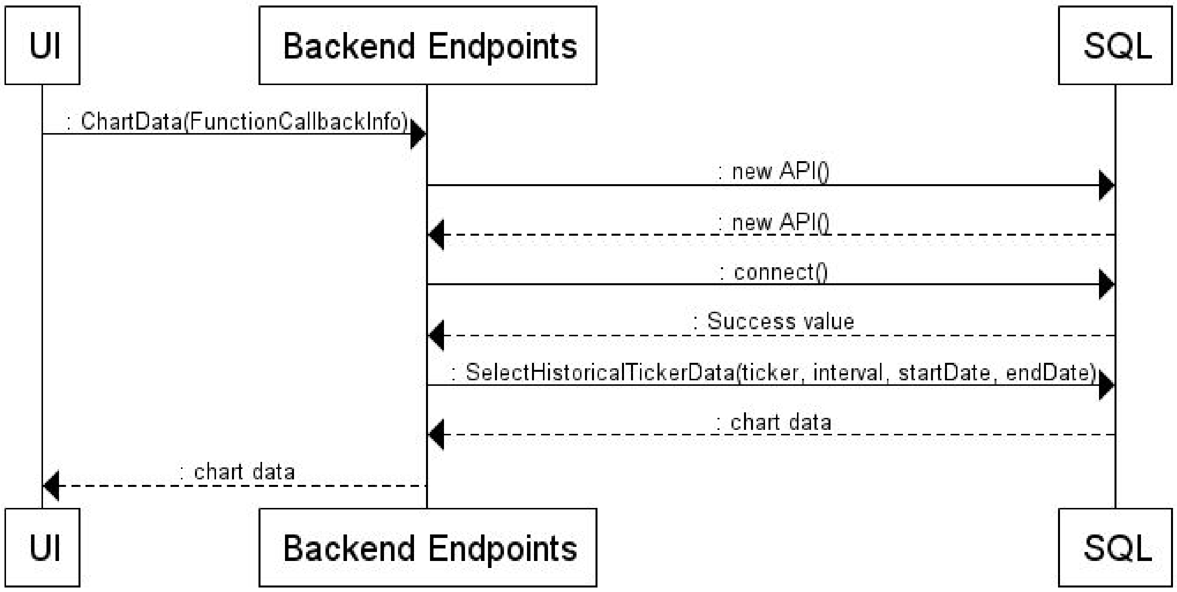
Sequence Diagrams:

Use Case #1
Finding optimal exchange path between currencies

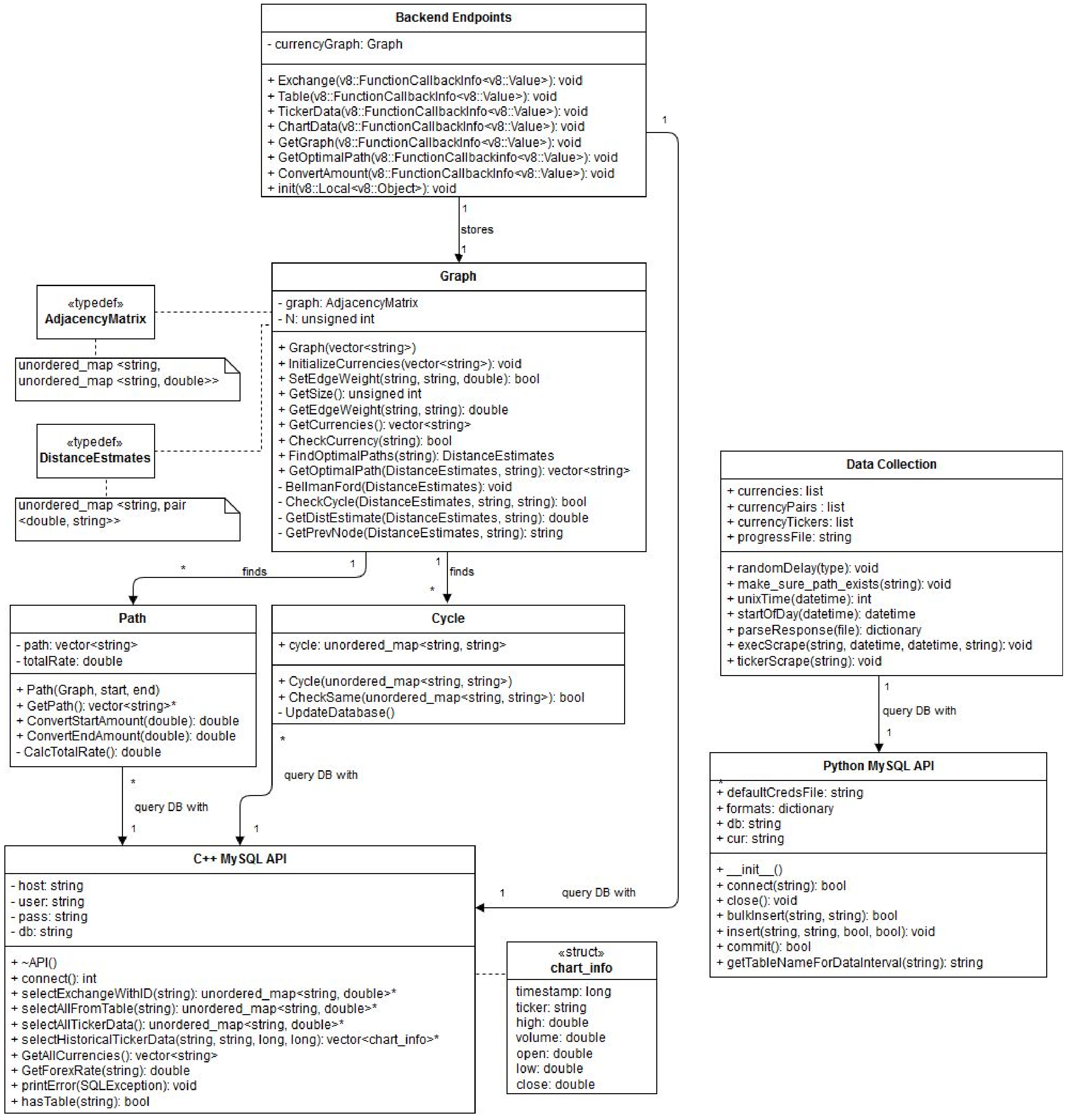


Use Case #2

Using Forex Dashboard



Static Class Diagram:


Backend Endpoints
- currencyGraph: Graph
+ Exchange(v8::FunctionCallbackInfo<v8::Value>): void
+ Table(v8::FunctionCallbackInfo<v8::Value>): void
+ TickerData(v8::FunctionCallbackInfo<v8::Value>): void
+ ChartData(v8::FunctionCallbackInfo<v8::Value>): void
+ GetGraph(v8::FunctionCallbackInfo<v8::Value>): void
+ GetOptimalPath(v8::FunctionCallbackinfo<v8::Value>): void
+ ConvertAmount(v8::FunctionCallbackInfo<v8::Value>): void
+ init(v8::Local<v8::Object>): void
1
stores
1
1
Graph
- graph: AdjacencyMatrix
- N: unsigned int
+ Graph(vector<string>)
+ InitializeCurrencies(vector<string>): void
+ SetEdgeWeight(string, string, double): bool
+ GetSize(): unsigned int
+ GetEdgeWeight(string, string): double
+ GetCurrencies(): vector<string>
+ CheckCurrency(string): bool
+ FindOptimalPaths(string): DistanceEstimates
+ GetOptimalPath(DistanceEstimates, string): vector<string>
- BellmanFord(DistanceEstimates): void
- CheckCycle(DistanceEstimates, string, string): bool
- GetDistEstimate(DistanceEstimates, string): double
- GetPrevNode(DistanceEstimates, string): string
«typedef»
AdjacencyMatrix
unordered_map <string, unordered_map <string, double>>
«typedef»
DistanceEstmates
unordered_map <string, pair <double, string>>
*
finds
1
1
finds
*
Path
- path: vector<string>
- totalRate: double
+ Path(Graph, start, end)
+ GetPath(): vector<string>*
+ ConvertStartAmount(double): double
+ ConvertEndAmount(double): double
- CalcTotalRate(): double
Cycle
+ cycle: unordered_map<string, string>
+ Cycle(unordered_map<string, string>)
+ CheckSame(unordered_map<string, string>): bool
- UpdateDatabase()
*
query DB with
1
*
query DB with
1
C++ MySQL API
- host: string
- user: string
- pass: string
- db: string
+ ~API()
+ connect(): int
+ selectExchangeWithID(string): unordered_map<string, double>*
+ selectAllFromTable(string): unordered_map<string, double>*
+ selectAllTickerData(): unordered_map<string, double>*
+ selectHistoricalTickerData(string, string, long, long): vector<chart_info>*
+ GetAllCurrencies(): vector<string>
+ GetForexRate(string): double
+ printError(SQLException): void
+ hasTable(string): bool
1
query DB with
«struct»
chart_info
timestamp: long
ticker: string
high: double
volume: double
open: double
low: double
close: double
Data Collection
+ currencies: list
+ currencyPairs : list
+ currencyTickers: list
+ progressFile: string
+ randomDelay(type): void
+ make_sure_path_exists(string): void
+ unixTime(datetime): int
+ startOfDay(datetime): datetime
+ parseResponse(file): dictionary
+ execScrape(string, datetime, datetime, string): void
+ tickerScrape(string): void
1
query DB with
1
Python MySQL API
+ defaultCredsFile: string
+ formats: dictionary
+ db: string
+ cur: string
+ __init__()
+ connect(string): bool
+ close(): void
+ bulkInsert(string, string): bool
+ insert(string, string, bool, bool): void
+ commit(): bool
+ getTableNameForDataInterval(string): string

CRC Cards:

The left side of each CRC card describes the class, its contents, and its responsibilities.
The right side rates the Cohesion of the class, and lists collaborators and their Coupling rating.
Cohesion and Coupling scales are as follows, from worst to best:

1. Coincidental Cohesion
2. Logical Cohesion
3. Temporal Cohesion
4. Procedural Cohesion
5. Communication Cohesion
6. Functional Cohesion
7. Informational Cohesion

1. Content Coupling
2. Common Coupling
3. Control Coupling
4. Stamp Coupling
5. Data Coupling

C++ MySQL API ● Contains: ○ SQL database credentials, driver ● Responsibility: ○ Performs SQL queries on database	Informational Cohesion

Python MySQL API ● Contains: ○ SQL database credentials, driver ● Responsibility: ○ Perform SQL queries to enter and update data on the database	Functional Cohesion ● **Coupling Ratings** ○ Data Collection: 5

Backend Endpoints ● Contains: ○ A Graph object ● Responsibility: ○ Execute C++ code requested by Node	Logical Cohesion ● **Coupling Ratings** ○ C++ MySQL API: 5 ○ Graph: 4

Graph ● Contains: ○ Graph of currencies and rates as an adjacency matrix ● Responsibilities: ○ Perform shortest path algorithm from a starting node ○ Identify positive cycles	Functional Cohesion ● **Coupling Ratings** ○ Cycle: 4

Path ● Contains: ○ List of currencies that form an exchange path ○ Combined rate from start to finish ● Responsibility: ○ Calculates currency exchanges along path	Functional Cohesion ● **Coupling Ratings** ○ Graph: 4 ○ C++ MySQL API: 5

Cycle ● Contains: ○ List of currencies that form a positive cycle of exchanges ● Responsibility: ○ Updates database with cycle information	Functional Cohesion ● **Coupling Ratings** ○ C++ MySQL API: 5

Data Collection ● Contains: ○ Requests to online APIs and web scrapers ● Responsibility: ○ Collect currency exchange data	Logical Cohesion ● **Coupling Ratings** ○ Python MySQL API: 5

Design Approach:

CurrencyHFT's design emphasizes speed, reliability and accessibility. Early on, the team decided to implement CurrencyHFT as a web application. This decision is consistent with industry trends as more applications are made available via the web. Web applications boast increased reliability and accessibility compared to desktop applications. CurrencyHFT is a research oriented application, thus users are likely to favor a web application over a mobile application, further solidifying the decision to develop a web app.

CurrencyHFT's front-end is implemented as a client rendered single-page application (SPA). This type of web application has rapidly grown in popularity, because it provides a user experience similar to that of a desktop application. Client-side rendering utilizes JavaScript to produce HTML and manipulate the Document Object Model (DOM). Client-side rendering utilizes the observer design pattern to asynchronously respond to user events. This approach allows the webpage to respond immediately to user events, rather than waiting for the server to send down HTML. Single-page applications utilize client-side rendering to greatly increase responsiveness at the cost of slower startup time and increased client resource usage. In most cases, including CurrencyHFT's, this tradeoff is favorable.

The SPA is served using a Node.js web server and implemented in the vue.js framework. Vue was chosen over other leading front-end frameworks for a host of reasons. Vue allows for modularity within the codebases. Reusable bits of the application can be broken into small elements, each with a consistent structure following the same designated logic. Further, unlike other frameworks, every bit of Vue is its own component, nested within parent components, all nested in the overarching App component. These components allow for a more modular design of the web application, which decreases coupling. The Vue router itself is a component, leading to another benefit of Vue - limited peer dependencies. Vue comes prepackaged with its own router (Vue-router) and state container (Vuex). These two modules streamline the entire development setup process upon the initial install. This reduces startup time and diminishes the learning curve, all in all allowing developers to build a frontend application fast. With an ever evolving ecosystem for support and very thorough and well organized documentation, Vue is an excellent choice.

The team separated the front-end and back-end by utilizing a RESTful API. The Node.js web server defines data endpoints which enable the front-end to request JSON data via AJAX calls. This setup was chosen for ease of implementation and to again emphasize responsiveness, due to AJAX's asynchronous behavior.

Node.js these queries, but functions only as a lightweight web server wrapper. For further computation, the endpoint logic is executed in C++, for its speed and object-oriented nature. Speed is especially advantageous for CurrencyHFT. The BackendEndpoints class defines the behavior of the web server endpoints. For data requests (like those on the Forex Dashboard), this class interacts directly with an API class to request data from a MySQL database. MySQL stores the data needed for the arbitrage algorithms and was chosen for its speed and relational format. For computational requests (like those on the Arbitrage Page), the BackendEndpoints class first interacts with Graph class. The Graph class stores a graph in an adjacency matrix, chosen for its improved runtime for dense graphs. The graph class defines methods for calculating a Bellman-Ford shortest path algorithm. The Bellman-Ford algorithm was chosen over Dijkstra’s because the exchange graph contains negative edge weights. Bellman-Ford’s shortest path has a runtime of O(|E||V|), which was known at the outset of the project and contributed to the team’s choice of C++. The algorithm makes use of the Path and Cycle classes during computation. These classes reduce coupling of the algorithm and allow for cleaner code.

Lastly, CurrecyHFT’s Forex market data ingestion is also object oriented and written in Python. Python was chosen over other languages for its ease of implementation and abundance of useful libraries. The DataCollection class pulls data from a Yahoo API and stores it to disk. Additionally, the class uses the DBAPI class (similar to C++’s API) to write data directly into the MySQL database. The python libraries, Node.js endpoints, and two MySQL API classes are all implementations of the Facade design pattern.

Overall, the team made design choices based on CurrencyHFT’s specific needs. C++ is used for the back-end to mitigate the risks of slow Bellman-Ford runtimes. The Facade design pattern is used to decrease coupling and enable team members to contribute in tandem. Node.js and Vue.js are used to build a single-page application that imitates the user interface of a desktop application, but with the accessibility of a web application. The Observer design pattern is used to support asynchronous interaction with the front-end. This well defined design approach will pay dividends throughout the construction phase.

Contribution Summary:

All team members contributed to editing, reviewing, and approving the Stakeholder Review 1 deliverable for final submission. Breakdown of individual contributions is as follows:

<u>Jack Cusick</u>
Design Approach

<u>James Nakashian</u>
Static Class Diagram, CRC Cards for C++ MySQL API, Python MySQL API, Backend Endpoints, Graph, Path, Cycle

<u>Patrick Engelsman</u>
Use case #1 sequence diagrams, CRC Cards for Python MySQL API, Data Collection, Backend Endpoints

<u>Matthew Cordone</u>
Use Case #2 sequence diagram, Status Report, Backend Endpoints

<u>Ayushi Mishra</u>
Design approach, status report

Status Report:

Up to this point, the team has completed the fundamental features needed for the application. The main purpose of CurrencyHFT is to find arbitrage opportunities in the currency exchange markets and to find the optimal exchange paths between two currencies. We have been developing, documenting and tracking the branch management system on GitHub.

We currently using day by day exchange rates between the top 8 currencies traded in the Forex market. In our upcoming iterations we will be importing the exchange rates between these currencies through multiple different banks and intermediaries.

We are currently running a web server using Node.js. We have implemented a Node to C++ connection in order to link the server to the main body of the code. We have built a Python API web-scraper which pulls the Forex data from online financial sites. The data collected is accessible via a C++ API which connects to our MySQL database which stores the market information.

We have created a shortest path algorithm using the Bellman-Ford process which takes the currency information we have stored and finds the optimal exchange path between two currencies depending on the user input. In our future iterations we will be expanding the algorithm to search through the currency rates between different banks and intermediaries.

We currently have a working front-end framework and dashboard. We are using Vue.js to create a SPA. We currently have a working dashboard and landing page, with a navigation bar and routing. The Dashboard displays historical Forex data in an interactive chart and major currency pairs.

One risk the team faces is retrieving bank exchange rates. It is often difficult to access bank exchange information programmatically.

The team is on track to complete CurrencyHFT by the deadline. The schedule has been continuously updated and the team is currently in iteration 4.