Planning and Design

Group 28

January 2020

Preface

This document is intended as a reference for the development team to develop a system that meets the requirements specified in the requirements analysis document. It can also be viewed by the client to gain a better understanding of how the software will be developed. This document details:

- An overview of the system and its structure,
- Design and implementation of different components of the system,
- System verification and testing methods at a unit, module and holistic level,
- Organisation of project workflow and team member roles,
- Interacting with the client and delivery of the system.

System Overview

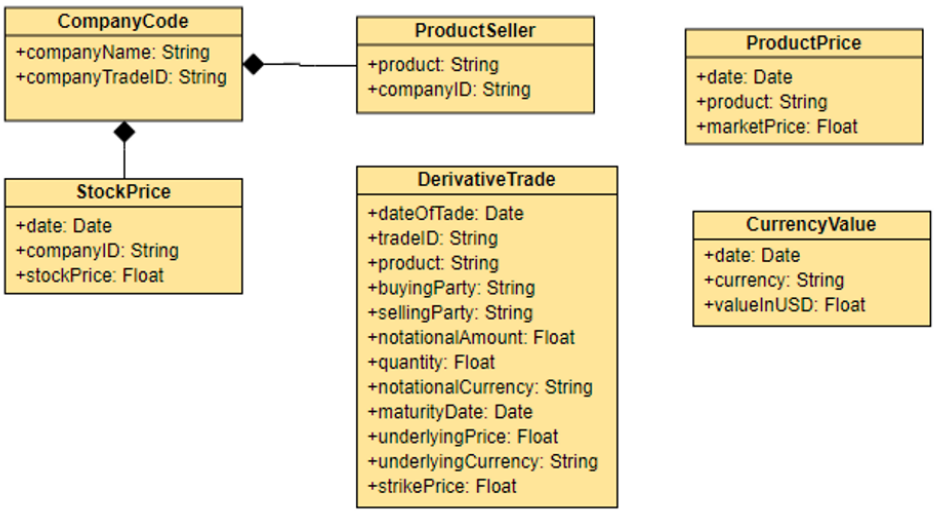
The software to be developed is a reporting system that allows users to enter and store details of derivative trades and related data. The system will learn to detect likely erroneous inputs based on past market and input values, flagging them to the user as possible errors. Daily reports will be generated based on the data in an appropriate format for the regulatory body to audit. The user will be able to view and correct past trades up until a week after the trade was submitted.

System Structure

We will implement the system using a client-server model, in which users will use a web-based interface to interact with a server. This type of model scales well with distribution, is well-suited to variable loads and allows access from multiple locations to a single database, making it extensible. The server will store the inputted data in a database and be responsible for generating and storing reports on a daily schedule. The server will also host the error detection module, which will use a statistical model based on previous values to evaluate new inputs, notifying the client when potentially erroneous values have been inputted.

Data Structures

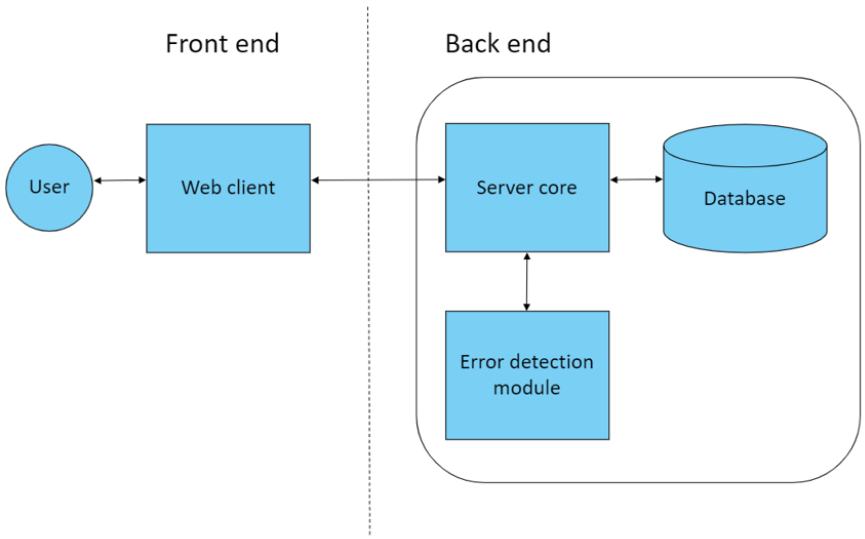
All parts of the system will use consistent data structures to represent and store information provided by the user. All data will be stored in a database. These data types are detailed below:

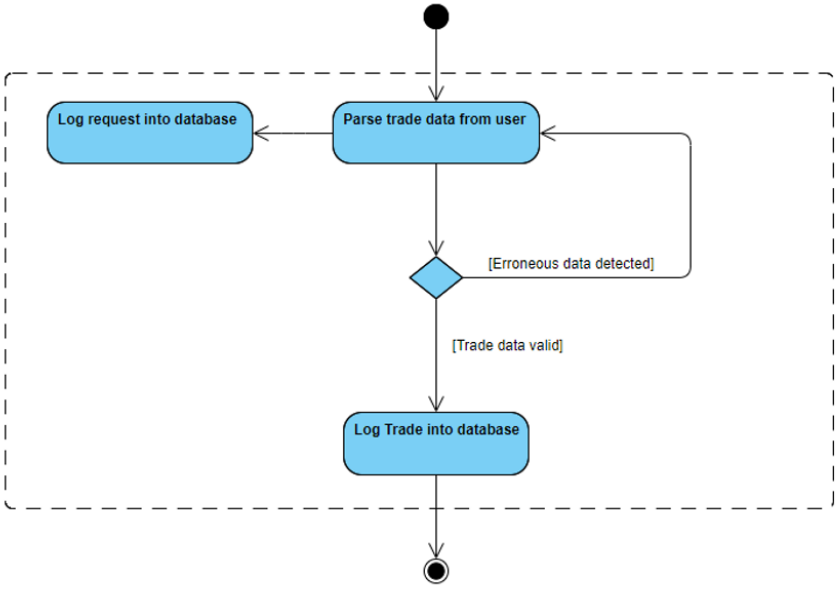


Implementation Details

Front End

The client software for our system will be implemented as a single-page web application using the JavaScript React framework, allowing users to access the system through a browser without installing additional software. React is a widely used framework for creating robust and structured applications, with support and documentation readily available. Combining React with Node Package Manager provides additional modularity and enables the use of existing libraries that provide required functionality or can be used to extend the system. Also, by dividing the page into separate “components”, individual elements can be reused, maintaining consistent design and allowing for easy maintenance of code. The system will be accessible from a variety of devices and platforms [D1.1]. The front end will use a REST API to receive and send data to and from the back end. Specifically, the details of new trades as well as edits to existing trades will be collected as user input and sent to the database, with the back end providing data concerning error-checking and existing trades. User input will be sanitised before it is sent to the back end to prevent malicious input, such as SQL injections. The REST API is lightweight with widespread usage and an appropriate data exchange protocol for this system architecture.





Back End

The server for our system will be implemented in Python, using the Django web framework, making it compatible with any system that supports Python. This is due to the vast selection of libraries that exist for both data analysis and web development which support both the front end and error detection development, allowing for code reuse and extensibility should additional data dimensions be introduced. Python also has a shallow learning curve due its readability and user-friendly data structures meaning all team members can pick it up fast, even without much previous experience. This versatility of Python leads to rapid development as one code base can be used (written in one language), rather than several code bases, each in different languages, being developed in parallel. Django is widely used and extremely easy to set up and run. It's interface for dealing with the database requires little to no SQL and tables can be modelled and built in minutes.
Django web framework will be used in conjunction with a Postgres database. The system will be running on CentOS Linux, using nginx as the HTTP server. Gunicorn will be our Web Server Gateway Interface (WSGI) application server used to interface between our Django app and the HTTP server. The app will mainly act as a REST API to the front end, handling requests to and from the user, executing processes when necessary to maintain reliability. A daily cron job running on the system OS will generate and save the report for the regulatory body [C8]. The server will call upon the error detection module when needed in the request-response cycle. As the error detection module is separate from the request-handling components the system should remain active if the module fails, i.e the user is able to browse and delete trades however creation and editing may become temporarily unavailable.

Error Detection and Correction

The server will host an error detection and correction module which will use the trade data in the database to build a statistical model of typical correct and incorrect values for various fields. When new data is entered by a user, it will be compared against the model to see how well it fits the pattern of the existing data. If the new data point has a high probability of being an outlier, the model can be used to propose a more likely value, which will be sent to the front end to suggest to the user [C2]. Additionally, more basic checks will be done to check for typos e.g if "UDS" is entered instead of "USD".
Deciding which modelling technique to use will require thorough analysis of the provided sample data, to deduce the structure of the data used in the system. A variety of Python libraries such as SciPy exist to implement common statistical techniques in an efficient manner, which will likely be employed in our system.

Components

To aid in the speed of development, we have split the implementation of the system into several components, each of which is responsible for meeting a subset of the developer requirements.

Front End

- Trade entry page: A form that allows the user to enter the details of a derivative trade, or edit or delete an existing one. Will also implement the front-end feedback for the error detection system. Meets requirements D1.3, D1.4, D4.1, D4.2, D4.3, D5.1, D5.2, D5.3, D10.1, D14.1, D14.2.
- Trade table view: A table listing all trades in the database. Meets requirements D3.2, D3.3 and D7.3.
- Error detection feedback: Extensions to the trade entry form and table view that present error detection results to the user. Meets requirements D2.2, D2.3, D6.1 and D6.2.
- Report table view: A table listing all previously generated reports. Meets requirements D8.1, D8.2 and D9.1.
- Table view sorting and search: Interface elements added to the trade and report tables that allow the user to sort and filter the table view. Meets requirements D3.4, D3.5 and D8.5.
- Instructions: A page in the interface that explains to the user how to use the software. Meets requirements D11.1 and D11.2.

Back End

- Trade database: Server-side database that will store the trade information entered by the user. Meets requirements D3.1 and D4.4.
- REST API: Used to transfer data between the client and server. Does not directly meet any requirements but is necessary for the system to work.
- Report generator: Module in the back end responsible for generating PDF reports from past trades in the database. Meets requirements D8.3 and D8.4.
- Error detection module: Back end module responsible for detecting possible errors in entered data based on past inputs and suggesting sensible corrected values. Meets requirements D2.1 and D7.1.

User Interface

The front end of the client software will provide a clear graphical interface so users can interact with the system intuitively. Interface design is inspired by existing software used in the financial industry, such as *Bloomberg Terminal, Sharekhan Tradetiger, and CME Direct*. This is done in order to create a familiar and appropriate experience for a non-technical user [C11]. Appendices A, B and C show initial designs for different elements of the user interface.

Appendix A is a visual prototype of the window the user sees upon wanting to create a new derivative trade entry [C1]. A visual hierarchy of elements present in the window indicate to the user that the primary action of the window is to input data for each of the derivative attribute fields. Grouping the fields using white space introduces organisation in order to reduce clutter and spread of the primary function across a large space. To reduce complexity, utilities for filling in dates and times should initially be hidden, and only shown when input to the respective fields is being performed.

Appendix B displays an error pop-up message the user may see when the error detection module has found potentially erroneous values [C2]. The importance of error checking makes it vital that the interface is easy to navigate and makes errors obvious to the user. To accomplish this, the same navigation buttons will be present on all pages of the interface to provide a consistent design. Bold colours will be utilised to highlight errors. Additionally, the use of ReactJS and its virtual DOM means that changes to the back end database can be reflected in the front end quickly [C12, C13].

Appendix C prototypes the view presented initially to the user [C3]. A complex interface is avoided by distributing functionality across multiple views and tabs, with each individual view or tab providing reduced or hidden functionality. When combined the views offer all functionality required by the system without penalty – the layout is

designed to indicate causality, and so the user can directly manipulate what they expect the system to do. The primary functionality of viewing and accessing the table is supported by multiple secondary functionalities that may appear when the user focuses elements with mouse input – the aim is to reinforce the users predictions on the behaviour of the system by displaying options the user is likely to be searching for when bringing an element into focus.
During the development of the project, clarification on the users' expectations of system workflow will be used to produce user stories to help inform design decisions surrounding specific UI features, so while the main interface design principles supporting initial user requirements will be kept the same to maintain consistency, more specific minor details can be adjusted to tailor the user experience.

Testing

It is important to periodically test that the software works as intended over the course of its development. We will use automated tests every time changes are merged into the master branch in the Git repository, as well as at the end of the project to ensure the final system passes the tests. Unit tests will be used to test individual components of the system, and integration tests will be used to ensure that the system works as a unified whole [C15]. A stress test or soaking test will be used to simulate real conditions where the system is under load for long periods of time, to ensure reliability and robustness.

Front End

The front end will be verified by using automated tools to ensure that the client software responds as expected to user interaction and messages sent from the server. Selenium WebDriver will be used to automate user interaction within a real web browser, and a dummy back end server will be implemented to send test data to the front-end during testing. The graphical interface will be constantly tested for ease of use and simplicity throughout the development process to ensure that the software can be used by non-technical users. Additionally, the React components can be tested with tools such as the Jest JavaScript test runner, allowing for access to the Document Object Model and ensuring correctness.

Back End

The server can be tested by sending mock requests to it from a dummy front-end using Postman, a HTTP testing client. The server will be tested upon a dummy database, so that the environment the server is tested in is controlled. Mock SQL attacks can also be conducted here to maintain security in the back end. We can check that the server emits the correct response to the front end and makes the expected changes to the database during testing, in order to validate its behaviour. We will also test the integration between the web server and the error detection module, making sure they are consistent in parameters and return types. Unit tests will be used on all methods to make sure that they work correctly. We will make it a priority to have as close as possible to 100% test coverage on the web server.

Error Detection Module

The error detection module is harder to test in isolation, as its outputs are somewhat subjective. We will test it by generating sample input data that includes both correct and erroneous samples. This data set will be provided to the module, and the percentage accuracy of its output can be computed and used to evaluate its performance.

Integration Tests

To ensure robustness and reliability, testing of the entire system will be done by loading a dummy database into the back end and simulating user interaction using Selenium WebDriver. We will test if the expected modifications are made to the database, and the front end remains responsive.

Risk Analysis

It is important to be aware of issues that may arise during the project, so that they can be mitigated. Some possible issues that may arise are listed below.

Risk	Likelihood	Severity	Precaution
Team member ill or otherwise unable to work	Medium	Medium	Ensure no one team member is fully responsible for any part of the project. Make the rest of the team aware of any inability to work, and delegate to someone else.
Loss of project data	Low	High	Store several copies of important data and code, both on developer's local machines and in cloud storage. Inform all team members of proper usage of version control using Git to avoid accidental deletion or modification.
Copyright infringement	Low	High	Always check the license of third-party software used by our project. Follow the license terms as needed. Never use software with ambiguous or non-existent licensing. Do not use illegally obtained/pirated software.
Plagiarism – team members may copy concepts, code or documentation from other groups without informing the rest of the team.	Medium	Critical	Pay attention to suspiciously high output or a change in style of work produced by other team members and enquire about the source of such work. If plagiarism is detected report this immediately to the module organisers to avoid the entire team being penalised.
Exceeding deadline dates	High	High	Set milestones to be completed by certain deadlines over the course of the project, to ensure consistent pace. Prioritise the development of certain features over others as per the requirements analysis, to ensure the critical parts of the system are implemented.

Documentation

Over the course of the project we will produce documentation for the software, aimed at helping users to use the system and developers to work on it [C11]. The user documentation will explain usage of the web interface in clear, non-technical language for end users. The REST API documentation will detail the exact message formats that can be sent between the client and server, to ensure both development teams are working with the same interface. To make the system portable we will also create documents that explain the installation process to the client, and the build and testing procedures to developers.

Team Structure

Given the small size of the group, the team is organised in an informal, flat structure with development roles and high-level project roles distributed between all members.
Development roles were initially split into 3 sub-teams dedicated to front-end interface development, back-end storage development and error detection module development. All team members play a role in development as members are confident at applying technical skills and generally have some experience in developing software, either through professional work or personal projects. Movement and communication between sub-teams is flexible to encourage group cohesiveness, information exchange and help achieve workload balance. Technical decision making is done at both sub-team level with regards to the development area, with more critical decisions being made by reaching consensus amongst all team members.
Additionally, responsibilities for higher-level aspects of the project are given to team members based on personal preference, interest and skill sets in an attempt to maintain motivation and upkeep the business-oriented processes of the project.

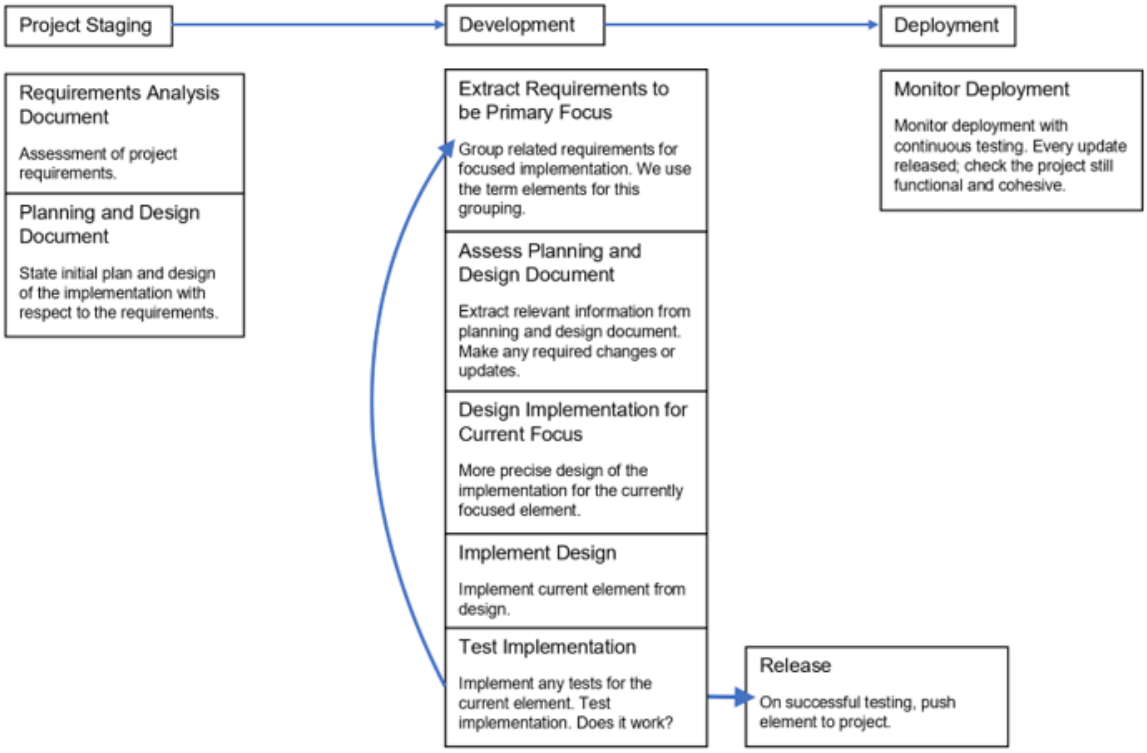
Communication in the group is done through informal weekly face-to-face meetings or through conversations in the workplace (University of Warwick), to facilitate common understanding and ensure a two-way flow of information. Additional non face-to-face channels are also used to accommodate for the generally distributed locations of team members and unavailability during weekly meetings:

- *WhatsApp*, for quickly resolving issues and general discussion.
- *Slack*, for directly related and sub-team specific communication and discussion of feeds from other tools such as *GitHub* and *Trello*.
- *Trello*, for enumerating and allocating tasks to team members in a Kanban-style system.
- *Microsoft Teams*, for sharing, storing and collaborating on documents.

Role	**Responsibilities**	**Team Members**
Front end development	Developing the front-end web-based client interface.	Artemy Bulavin, Yalun Zhang
Back end development	Developing the back end server and database.	Max Wilkinson, Michael Tyler
Error Detection development	Analysing sample data to determine the best method to use for error detection and correction and implementing an error detection module.	Sol Franklin, Mihai Iulian Cioroianu
Project Manager	Assigning work to the other team members, and project-level decision making.	Artemy Bulavin
Business Analyst	Evaluating the extent to which the planned solution will meet the needs of the client and ensuring that these needs are met.	Mihai Iulian Cioroianu
Development Operations Engineer	Setting up and maintaining the tools used by the team to manage the development process.	Max Wilkinson
Head of testing	Overseeing the proper testing of all parts of the system, including working with the development teams to set up test environments for each component of the software.	Sol Franklin

Development Life-cycle

We will develop the software using a combination of plan-driven and agile workflow. We will start with the basic high-level plan of the whole system specified in this document, which will guide the progress of the project. Over the course of the project we will plan more specific parts of the software in more detail, and then implement them before testing and evaluating our changes and moving on to the next part. We will plan the unit tests for each component before implementing it, to enable test driven development. This workflow should afford us the flexibility to meet changing requirements, whilst also thinking about exactly what each component will do before implementing it.

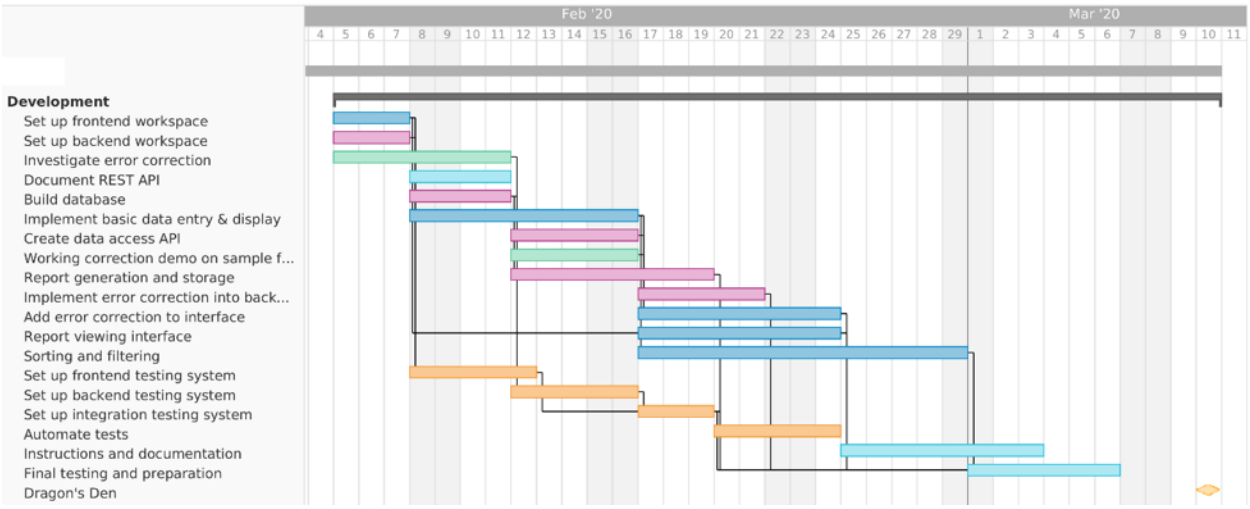


Development Tools

We will store our code in a Git repository hosted on GitHub. Each team member will use a separate branch to work on their tasks, regularly committing their changes with clear messages that will make it easy to roll back unwanted changes. Once a feature has been completed, a pull request will be made and reviewed by the DevOps engineer. At this point GitHub Actions can be used to run the automated tests described in the Testing section to automatically reject code that fails the tests. Once accepted, the changes will be merged into the master branch. Postman will be used to send dummy HTTP requests and responses for testing purposes so that the back end can be developed independently of the front end. We are using Trello to assign tasks to team members, and Microsoft Teams to share files and discuss the project. All non-code files pertaining to the project will be stored and worked on in Teams where possible, to ensure every team member is able to see all work undertaken within the project and avoid doing the same work twice. Communication has been done over Slack. We used this since some of the other apps mentioned above integrate into the channels meaning progress and work can be tracked from a central hub.

Project Timeline

The below chart gives a rough plan for the relative timings of various tasks within the project.

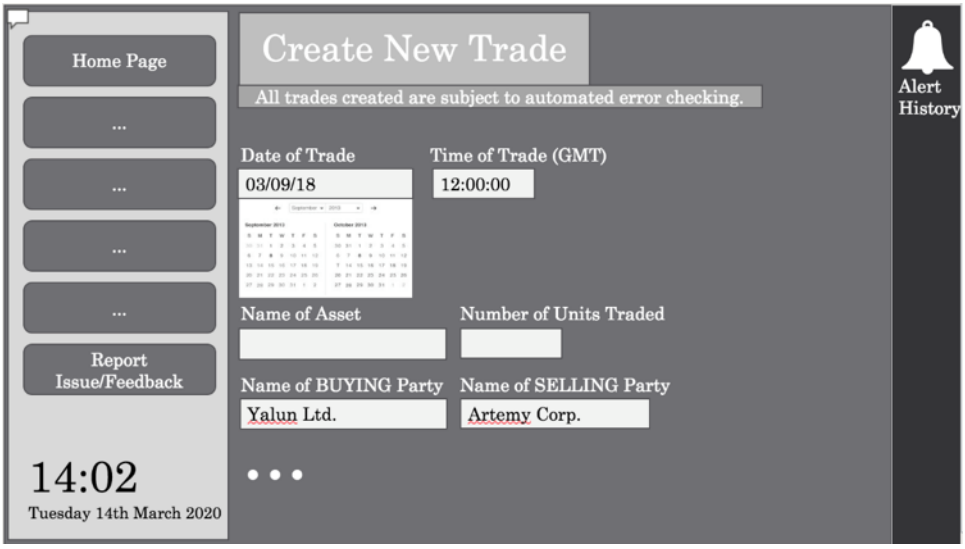


Delivery to Client

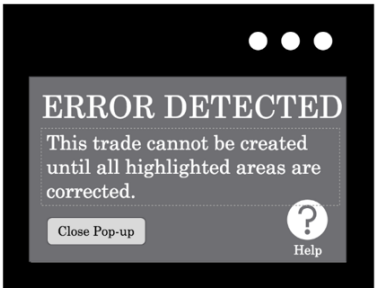
Communication with the client, Deutsche Bank, throughout the software development process is essential for a smooth delivery. In line with the project timeline and life-cycle, the finished software will be demonstrated to representatives of the client on 10th or 11th March 2020. The compatibility and portability of the system means they will have the opportunity to see the system in action and decide whether it meets their needs. A discussion about how the system meets each of the requirements set forth in the requirements analysis will take place, as well as a detailed demonstration of the tests used to validate the software. Due to the modularity of our design, changes to the front end and back end will be relatively easy to implement which shall be expressed to the client. Along with the final report, any potential queries from the client will be answered so that all stakeholders are satisfied with the product.

(Appendix on following page)

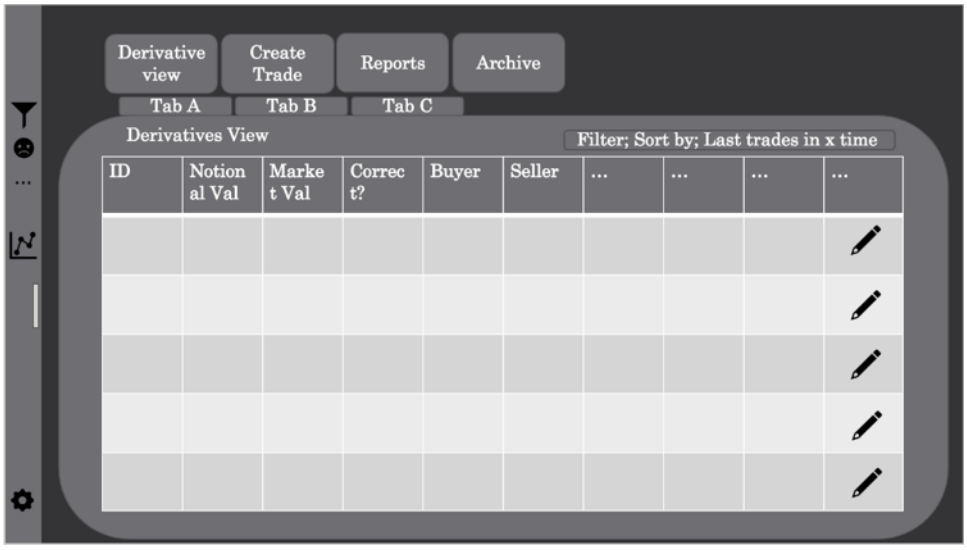
Appendix



A. User Interface Creating New Trade Page



B. User Interface Alert Pop-up



C. User Interface Main/Derivatives View