CS261 Final Report
Group 24

Stylianos Maimaris Tim Marshall-Cox Maciej Lulka Folabi Ayonrinde Josh Adams

Lewis Buttle

Contents

1 Introduction

The purpose of the Final Report is to provide information and insight into the CS261 Software Engineering Project for 2020 for Group 24. It has the intention of highlighting and outlining the implementation and evaluation of the project, with regards to the proposed plan in 'Requirements Analysis' and 'Design and Planning' reports. Encapsulated in the document are sections pertaining to: Research, Design Changes, Implementation, Testing, Project Management, Evaluation and an overall project conclusion. Each section is broken down into more focused areas to allow for more thorough discussion. The Final Report is targeted towards the client (Deutsche Bank) and the module organiser (Dr. James Archbold).

2 Changes to design

2.1 Front-End

Minor changes have been made to the design of the system. The process with which the user selects trades to edit or delete has been changed as it was deemed too complicated and did not provide enough functionality. The original design expected the user to input details of a trade which they intended on editing. This was altered to present the user with a list of trades available to be edited/deleted, significantly reducing the work required to locate a trade. Functionality to search and filter trades is also available to help the user find the desired result. Furthermore, the system will provide functionality for uploading data to the database in the form of CSV files. This change was made in order to help with the testing and demonstration of the prototype and may be removed in a future version of the system.

2.2 Back-End

The system's back end was largely completed in line with the design document, although there were a few notable changes. Firstly, the scikit python library was abandoned in favour of the SciPy [1] and NumPy [2] Python libraries. This was, for the most part, due to greater ease of use; it permitted the use of NumPy arrays which are very versatile and well documented online. This was addressed as a risk in the Risk Register. The mitigation ensured that the time lost was the time spent writing for scikit which only amounted to around 1 day.

In the design, the outlier thresholds were left undefined to allow them to be fine-tuned to the data. In the final application, anything within ± 2 standard deviations of the linear regressions model was treated as acceptable and coherent with the data. Anything between 2 and 3 standard deviations from the model was flagged to the user and run through the correction algorithm without replacing the user input automatically. Instead of applying potential changes, they were suggested to the user in a warning message. Any user input outside of 3 standard deviations was deemed erroneous and the correction algorithm attempted to correct it. If corrections were found, they were applied automatically. If no corrections were found, the user received a warning message stating that their entry was significantly different from historical data.

3 Implementation

The first step to implementing the system was to choose the structure of the web application. Django utilises 'applications'; pieces of code which are responsible for interacting with different parts of the framework [3]. Two separate Django applications were created: application for handling the front end and back end of the system and cs261_webapp which is used to set system settings such as URL patterns. The following sections aim to break down how each app is used for the implementation of the system and how that meets the project requirements. At the end of each subsection, a small conclusion is given in regards to which requirements were met as well as the overall development time required.

3.1 Front End

Having made the the changes to the user interface as stated in the Changes to design section, the implementation of the system followed the requirements and design stated in the Requirements, and Planning and Design documents respectively.

3.1.1 File structure

To ensure consistency throughout the system and meet the requirements for usability by staff with little to no training (Non-functional req. 2 and 3), the team decided to utilise Django's template inheritance system [4]. Specifically, an HTML template basic_template,html was created with the appropriate CSS styling which would act as the system's main menu. All other web pages within the system, including template pages provided by the Django framework, are set to inherit from it. This ensures that features such as the ability to access all system functions from everywhere in the system had to be coded only once. Furthermore, the visual design remains consistent on all pages, ensuring the user can easily navigate and use the system. The file structure used is shown in Figure 1.

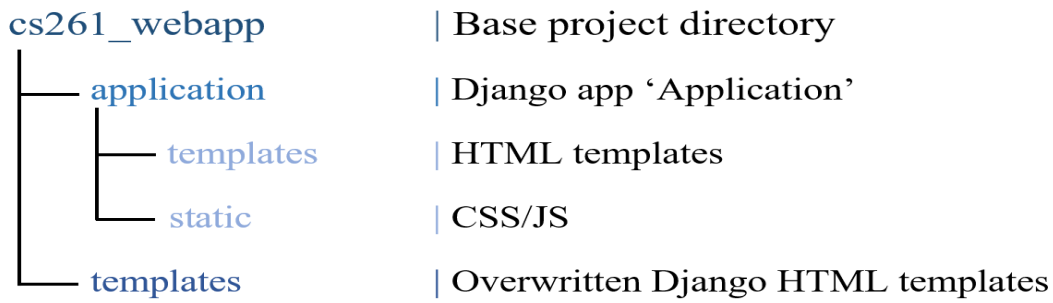

Figure 1: Front-end file structure

The file structure adheres to the Django standards ensuring that more functionality and web pages/views can be easily added. Furthermore the templates folder under the root directory allows for overwriting the existing templates provided by Django. This approach to implementing the front-end of the system allowed more head room for the team to experiment and proved to be very time efficient as the overall development required nine days as compared to the estimated eleven (project Gantt chart in Design and Planning report).

3.1.2 Main Menu

The main menu serves as the landing page for the system and provides access to all the main system functions. It inherits from basic_template.html to ensure visual consistency. In addition, it implements an HTML navigation bar at the top of the page which allows the user to access the creating, editing, and report generation features of the system from any page or the main menu by clicking 'Group 24'. This contributes to the ease of use of the system. The navigation bar is shown in Figure 2.

Figure 2: System-wide Navigation bar

The front-end team decided to use a grid based menu for the main content of the web page for a number of reasons. Firstly, such an approach allows for addition of more functionality very easily in the form of new buttons. Moreover, this approach is very common and easily understandable without the need for training as the buttons and their functionality are self described. Lastly, a button based grid translates very well to smaller screens such as tablets and mobile phones. This is important as the system can be accessed from such devices using a web browser. The main menu is shown in Figure 3. Each button redirects the user to separate Django view (detailed in sections 4.1.3 - 4.2.4) using the `HttpResponseRedirect()` function. A view in Django refers to either a Python class or function which is responsible for setting and rendering the content of a given web page or redirecting to one [5]. The main menu is displayed using the `MainView()` function in views.py which redirects to URL: HOST_URL/application. The HTML and CSS files used for the main menu are index.html (in application/templates) and index.css (in application/static).

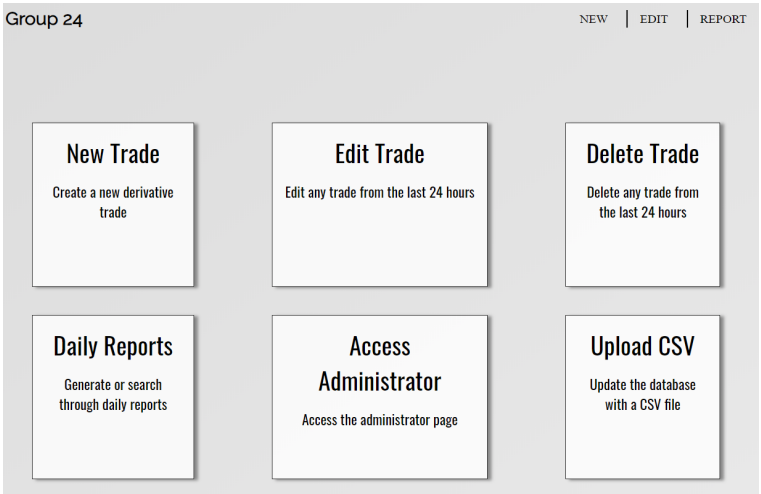

Figure 3: System main menu

EVALUATION: The development for the main menu was completed within the time frame set by the front-end team and meets the system requirements (Functional req. 1.D) while providing additional functionality in the form of the ability to access the utility view 'Upload CSV'. The styling was consistent with the rest of the site, and by creating a single base template ensured that future maintenance of the site would be made easier, as code to override all pages would only need to be altered once.

3.1.3 New Trade

The 'New Trade' view is the most complicated front-end aspect of the system and a number of features implemented here are also used by the edit and delete views. In addition, it is the view that has been iterated the most upon with two separate designs existing at the same time before one was scrapped in favour of the other. This was the case as the majority of interactions between the user and the system is creating or editing trades (which share the same input form) and as such it was deemed necessary to allocate extra time to ensure the end result was both easy to understand as well as easy to use. In addition to ease of use by the user, it was necessary that the form could be easily validated to be able to provide meaningful warning/error/success messages.

The first iteration of the view utilised a Django form designed by the front-end team to be similar to the `DerivativeTrades` model (Django models reflect database tables). Effectively, the user was presented with a form with basic data type validation for each field. Each field was read as the appropriate data type (Django form data types such as `forms.IntegerField`) and could present the user with error messages when data type violations were detected. This bare-bones approach was deemed to not be user friendly and required unnecessary manual labour by the user (filling fields such as date which could be autocompleted or buying/selling parties). As such this design was retired in favour of using the `ModelForm` feature of Django. This functionality of the framework allowed the front-end team to generate a form that mirrored the model one-to-one, enabled a number of additional validations, and provided quality-of-life improvements to the user. The final iteration of the form and 'New Trade' view are shown in Figure 4. The 'New Trade' view is displayed using the `CreateTradeView()` function in views.py which redirects to the URL: HOST_URL/username/application/derivativetrades/add/. The form class `DerivativeTradesForm` is defined in the admin.py file.

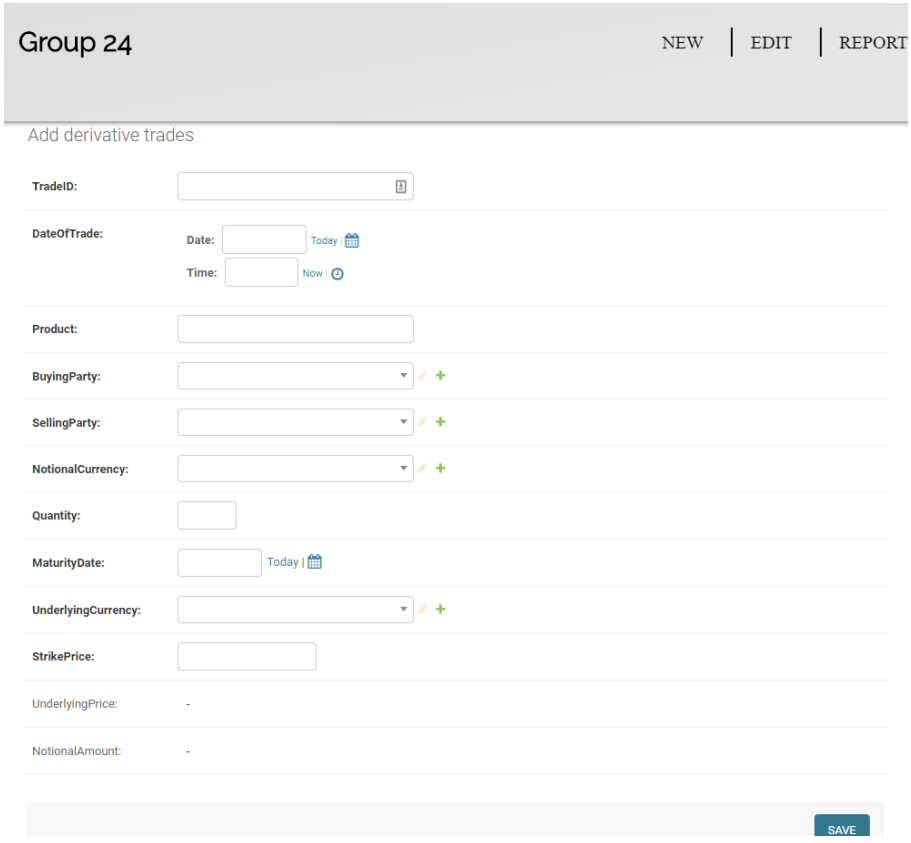

Figure 4: New Trade form and view

The user is presented with ten input fields (TradeID, DateOfTrade, Product, BuyingParty, SellingParty, NotionalCurrency, Quantity, MaturityDate, UnderlyingCurrency, StrikePrice) and two read only fields (UnderlyingPrice, NotionalAmount). The read only fields are calculated by the back-end after the user clicks the Save button and all other validation checks detect

no errors (either on front-end or back-end). The use of a `ModelForm` enabled the use of autofields. An autofield allows an input field of a given form to fetch data from an existing model as well as dynamically search for that data and be presented as a drop down list. This was used for the BuyingParty, SellingParty, NotionalCurrency and UnderlyingCurrency fields as shown in Figure 5 and 6. This approach to implementation eliminated any errors when selected the stated fields as the user is only able to choose from data that is supported by the system. Furthermore, the drop down lists in addition to dynamic searching minimise the time spent by the user filling out the form. Lastly, the user is able to add new companies or currencies on the fly as the form autofields are directly connected to their respective models in the database. This is shown in figures 7. New products and stock prices can be added through the administrator page (Section 4.1.8).

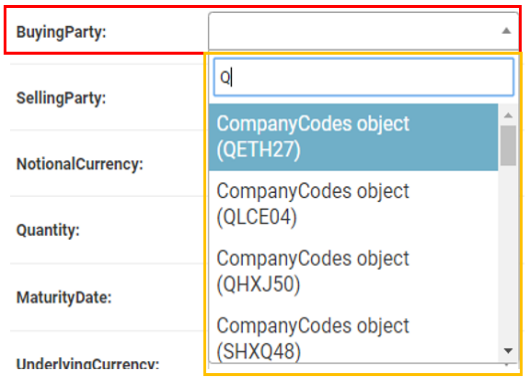

Figure 5: BuyingParty drop down list

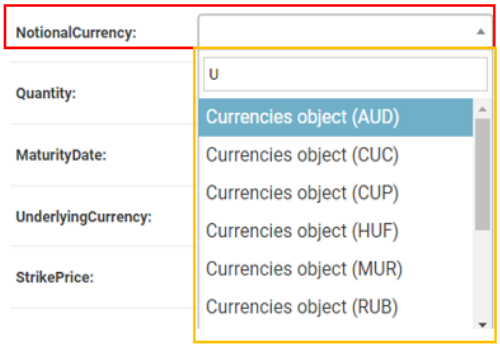

Figure 6: NotionalCurrency drop down list

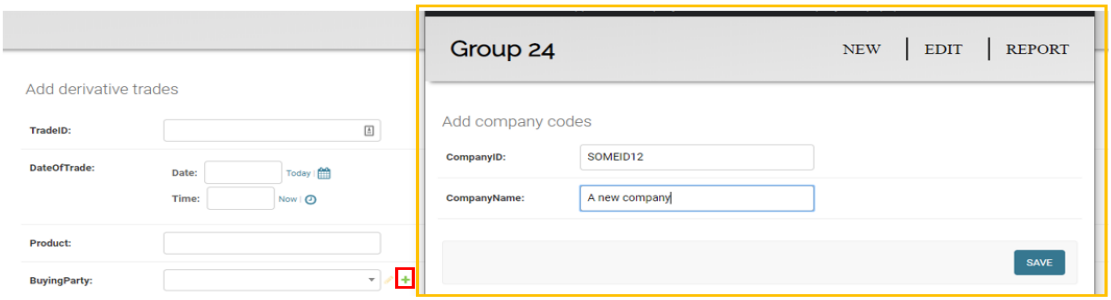

Figure 7: Add New Buying Party

In addition to automatic data type validation and autofield validation, additional validation systems were implemented. Before a user's input is passed to the back-end component of the system, each field was processed by implementing a `Clean()` function as part of the `ModelForm` (defined in the admin.py) class which is called whenever the Save button is clicked. This function performs all additional front-end validation. For the TradeID field, the choice was made to allow IDs of length 16, with the first 8 characters as letters and the last 8 as integers. This was done to match the provided test data. Moreover, the DateOfTrade field only allows dates that are between 2010 and the current date as no data for currency values, product prices and stock values exist outside that range. The MaturityDate field is checked to be greater than the DateOfTrade field. For the Product field, a smart suggestion system was designed using the Difflib Python library[6]. When a user enters the name of a product, that product is first checked against the list of available products sold by the selected selling party. If no such product exists, then the user is presented with an error message with suggestions for potential products sold by the selling party. This is achieved by querying the database (using a QuerySet, a Django Python set resulting from a database query) for objects in the `ProductSellers` model whose companyID field matches that of the selected selling party. In the event the product name input by the user is found to be similar to an existing product sold by the seller, then a suggestion is made to the user. This is implemented using the `get_close_matches function()` (part of Difflib) which returns the closest alphabetical matches to the given input which is the list of products sold by the seller as well as the user input for the Product field. This behaviour is shown in Figures 8 and 9. When the form validation is complete, a `DerivativeTrades` object is send to to the back-end component of the system for further processing.

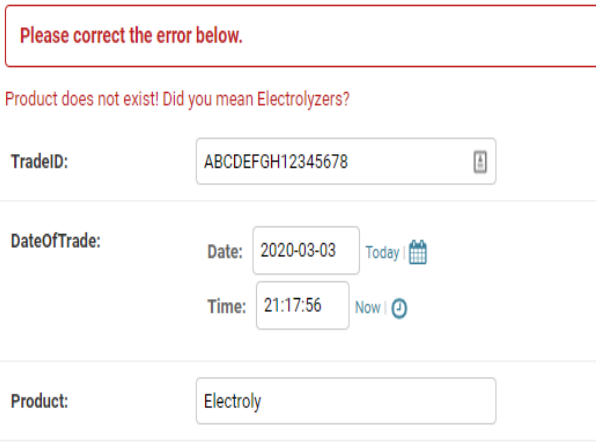

Figure 8: Product with typo error suggestion

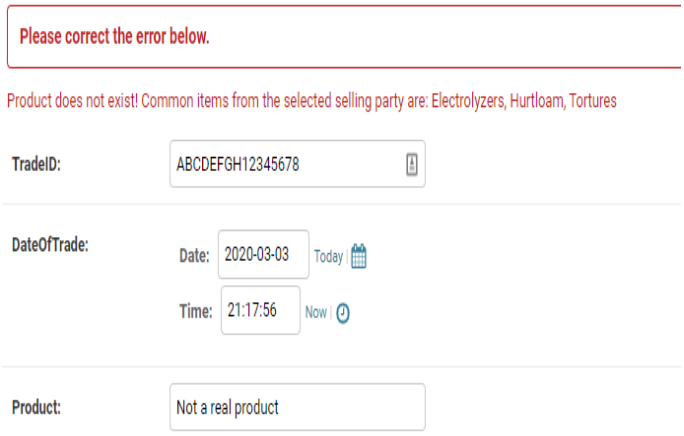

Figure 9: Product does not exist error with suggestions

EVALUATION: The overall development of the 'New Trade' view was significantly more difficult than anticipated. The original form design wasted nearly a day of development as almost none of the existing features were used during the redesign. Additional framework research would have been very helpful before starting the implementation process, as it would have ensured the team designed the solution in line with what was feasible. However, despite the day long set back, the new approach to the trade form ensured that the implementation for editing and deleting trades proceeded without unexpected issues, and even included additional functionality, such as the ability for a user to select a date from a calendar pop up window, and the time from a similarly designed clock icon. These changes reduced the time taken for an individual to fill in the form, increasing the usability of the site, and also reduced the ability of the user to make mistakes, such as by limiting their responses to certain fields to those only existing in the database. Therefore this final iteration of the form meets all requirements set by the team (Functional req. 2.1D-2.3D and 6.D)

3.1.4 Editing & Deleting Trades

The 'Edit Trade' and 'Delete Trade' pages share the same view which was designed by overwriting and integrating the functionality provided by the Django framework for existing database objects. Specifically, Django provides a bare-bones object viewer for any of the existing models (database tables). The front-end team, modified the object viewer according to the functional requirements for editing existing trades. As such, when a user clicks the 'Edit Trade' or 'Delete Trade' buttons (which execute the `EditTradeView()` and `DeleteTradeView()` functions from views.py), they are redirected to the URL: HOST_URL/username/application/derivativetrades/ where they are presented with list of trades which have been created in the last 24 hours. Any trades older than 24 hours fall outside the period within which a user is able to edit/delete them and they are not displayed. The list of objects is shown in 10.

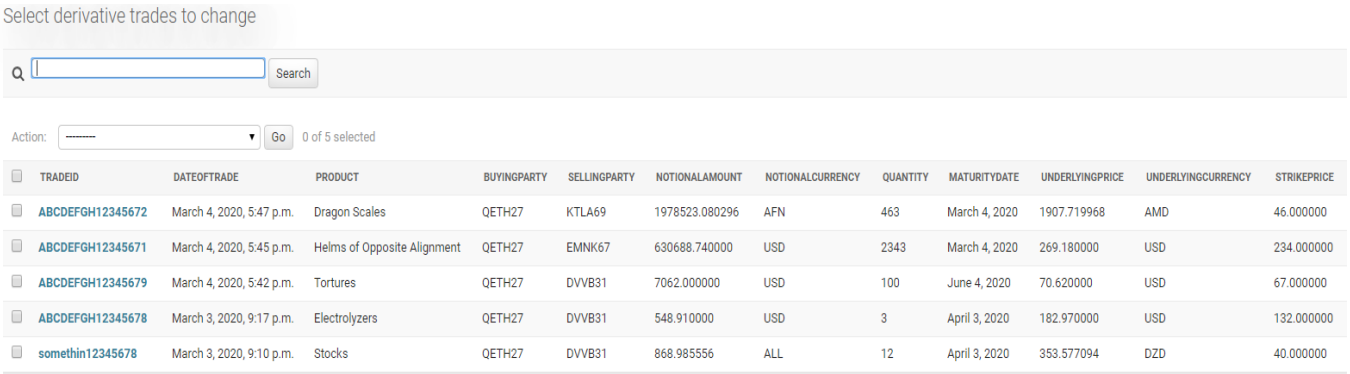

Figure 10: List of trades that can be edited/deleted

In order to limit the trades which are displayed to the user, the QuerySet for displaying trades was filtered in order to only show trades marked as archived. This is implemented in the admin.py file under the `DerivativeTradesAdmin` class using the `get_queryset()` function. Furthermore, a function called `updateArchivedTrades()` was written in the views.py file which, using the Datetime Python library, filters all trades which were created over 24 hours ago and updates the Archived field for each, setting it to True. This ensures that historic data is preserved and can be used by the back-end system as

well as for generating daily reports. Moreover, the search field enables a user to search for any trade by matching the search input to any of the fields displayed. In addition, each column shown in Figure 10 can be clicked, which results in alphabetical sorting of the list by chaining filters on the displayed QuerySet. Finally, the user is able to select multiple trades by clicking their respective tick box and delete them by selecting the Delete action. This is checked using the POST information when a user clicks any action on the web page.

When the user clicks on the tradeID of any of the trades, they are presented with the same model form with which they created the trade. The ensures that any data that is altered is validated before being saved, preventing erroneous data being entered into the database. The user is also able to delete the selected trade using the Delete button on the bottom left corner of the form, and there is additional functionality on this page to allow any user to view the history of changes to a single trade.

EVALUATION: The development of the editing and deleting functionality took place after the overhaul of the 'New Trade' view. As such, most research had been completed, and so there already existed code that inherited from Django admin, and so utilising this again significantly reduced development time and lead to better integration with the other system components. It also ensured that the styling of the web page was consistent with the previous, inheriting the navigation bar from the base template, and utilising the same form as when a user enters a new trade, except now filling the fields with the current data. Much of the code from new trade was reused, especially the validation, modularizing the solution. Testing now only had to be performed on the system once, as it was guaranteed to work across the two pages. This reduced the possibilities of errors, contributing to the maintainability of the site. Overall, the functionality on this page met the requirements specified (Functional req. 3.1D-3.3D and 4.1D-4.3D) and fulfilled them within the deadlines set.

3.1.5 Daily Reports

A major discussion point within the front-end team was how to handle the implementation of generating the PDF daily reports for trades. The team decided to have two different systems available to the user when requesting a daily report. Firstly, when the user clicks the 'Daily Reports' button in the main menu view, the view function `DailyReportView()` in views.py is called which performs a check on the current system time. If the time is within the range of 23:59:00 and 23:59:59, the report for the day is generated automatically. If however that is not the case, the user is presented with a choice form as shown in Figure 11. When the user clicks either option, a form with a date input field is displayed as shown in Figures 12 and 13. The system determines the choice of the user by inspecting the POST request data within `DailyReportView()` function.

When 'Generate New Report' is chosen and a date is given, the `DailyReportView()` creates a QuerySet of all trades that have been created on the given day. Next, the Reportlab Python library[7] is used to create a blank PDF document using the `SimpleDocTemplate()` function. The next step is the creation of a Reportlab table object with each row consisting of the data from a single entry in the retrieved QuerySet. This table is then appended to the PDF document along with a title in which the date of the report is stated. The PDF is then automatically displayed to the user and a copy of it is saved in the reports directory (cs261_webapp/reports). An example report is shown in Figure 14. When 'Search Through Reports' is selected, the system will instead search for the selected date within the reports sub directory under the root project directory. If a report for the selected date exists, it will automatically be displayed to the user, otherwise a message will prompt the user to generate the report instead.

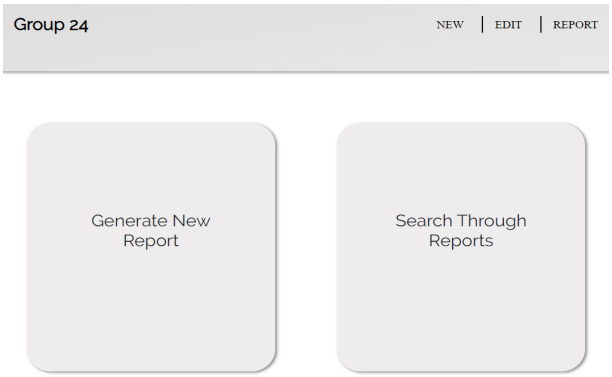

Figure 11: Daily Report Choice Form

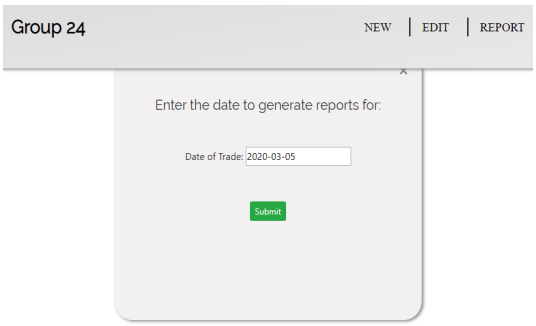

Figure 12: Daily Report Generate Form

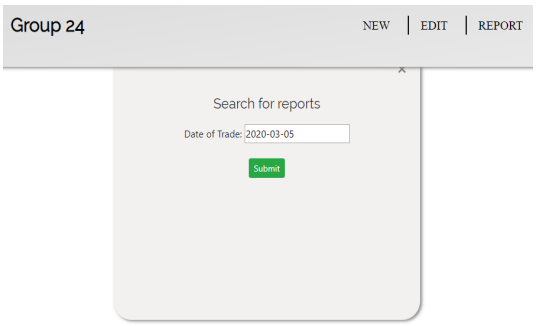

Figure 13: Daily Report Search Form

Daily Report for 2019-03-05

Trade ID	Date of Trade	Product	Buying Party	Selling Party	Notional Amount	Notional Currency	Quantity	Maturity Date	Underlying Price	Underlying Currency	Strike Price	Archived
OFMKTWZP68468218	2019-03-05 00:02	Amulets of Yendor	PKBF70	OPET16	1847722.46	KHR	5000	2020-11-05	227.51	USD	411.18	True
TMOVAQDC42891931	2019-03-05 00:02	Razor Fangs	KUWI70	YSSR29	1435.00	USD	100	2021-02-05	14.35	USD	16.00	True
DBLYVHOG69735886	2019-03-05 00:03	Stocks	NCXS33	OKYS51	18240.00	USD	4000	2020-05-15	4.56	USD	4.02	True
NOWZTZSM34693371	2019-03-05 00:03	Stocks	MFJJ99	VCSF07	3436.00	USD	100	2020-01-31	34.36	USD	31.71	True
XAUZVRQS85623777	2019-03-05 00:04	Stocks	TGZI54	QHXJ50	97622.00	USD	700	2022-06-04	464.72	HKD	162.35	True
MOLKRFVN63270973	2019-03-05 00:04	Stocks	NCXS33	DVVB31	43065.00	USD	500	2021-11-14	86.13	USD	83.70	True
YRRQSOZT64523809	2019-03-05 00:05	Stocks	FAWI50	OPZI26	32001.35	BTN	300	2021-03-05	15.89	USD	124.91	True
DAZBZVJX13343897	2019-03-05 00:05	Stocks	ESPL27	AGBH66	14235137.00	HNL	50000	2022-08-14	104.02	USD	266.49	True
REOTMACC97555944	2019-03-05 00:05	Armour Magnetizers	EMNK67	CUVZ50	35502.00	USD	200	2023-05-03	177.51	USD	178.97	True
TOJELZEH10645504	2019-03-05 00:05	Stocks	SXTM39	HFLM11	7054213.75	LBP	5000	2024-01-20	114.47	USD	1637.42	True
LWREEBBU65996964	2019-03-05 00:06	Stocks	CUVZ50	VKMV12	2757359.28	SZL	10000	2020-03-17	55.92	USD	266.03	True
IBKSEDRH40048865	2019-03-05 00:06	Z-crystals	DREA89	QBAP68	337740.00	USD	2000	2023-04-21	168.87	USD	183.14	True
EWOJKUXL32043215	2019-03-05 00:07	Ninja Info Cards	ZBIO05	EKXA24	939960.00	USD	6000	2021-10-02	156.66	USD	135.03	True
VRBEBVWA66412869	2019-03-05 00:07	Stocks	JXGF57	GIWG34	184382.64	PKR	8000	2024-09-12	1.95	USD	20.74	True
BFDCXUTU82999290	2019-03-05 00:07	Stocks	DVVB31	JVHD93	11190.58	NGN	7000	2023-01-11	12.47	USD	1.88	True
EIZCWISQ88912426	2019-03-05 00:08	Stocks	HXDA52	OPET16	3546.00	USD	900	2022-05-12	3.94	USD	4.36	True
XITXQDQL22809769	2019-03-05 00:08	Saibamen Seeds	RJCA92	YCZA59	180773042.52	UAH	8000	2021-10-28	723.59	USD	24398.20	True
TAFGXORU97728110	2019-03-05 00:10	Stocks	RJCA92	HFLM11	9157600.00	USD	80000	2023-06-17	114.47	USD	94.41	True
HYQEZDOE07136669	2019-03-05 00:10	Stocks	CMOI26	TKVF94	25455.00	USD	500	2024-09-13	50.91	USD	53.45	True
RGNAHBKL16919209	2019-03-05 00:11	Stocks	TBVE46	VGYT30	14742400.00	USD	80000	2021-02-21	184.28	USD	202.41	True
WEEBIUNG61464486	2019-03-05 00:11	Orcarinas of Time	VVBY18	PIFN94	118632.00	USD	600	2020-07-19	197.72	USD	218.27	True
FHIPLTXB55197676	2019-03-05 00:12	Gargantua Space Stations	INIL60	FYYD57	593000.00	USD	50000	2022-03-27	11.86	USD	13.81	True
ZFBLOLBJ38362957	2019-03-05 00:12	Astro Boys	HWJF09	VPTI00	660120.00	USD	300	2020-03-14	2200.40	USD	2387.68	True
OGCPNCLA80612279	2019-03-05 00:12	6IV Shiny Dittos	DNIL23	AMBT33	401589.00	USD	900	2023-02-23	446.21	USD	379.67	True
YYKMREIK74095415	2019-03-05 00:12	Golden Lucky Cats	ZFHD93	KRRO64	730378.68	MMK	800	2024-11-04	207.73	USD	819.10	True
LKLIHWDQ11672215	2019-03-05 00:12	Master Swords	XRBV61	ZVEN26	551533.14	TZS	2000	2024-01-27	117.02	ZAR	270.61	True
HECMUYSH20166479	2019-03-05 00:13	Surrounders	DILF10	TKVF94	657360.00	USD	8000	2021-02-21	82.17	USD	70.70	True
XMUPFCHN60299120	2019-03-05 00:13	Stocks	QBAP68	AYIH50	963080.00	USD	4000	2023-02-11	240.77	USD	285.91	True
ATEPIVNB87127917	2019-03-05 00:14	Dark Planet Cola	AMRO66	HWHS51	79898602.40	CAD	50000	2021-06-29	592.18	CNY	1715.68	True

Figure 14: Daily Report PDF Example

EVALUATION: Implementing report generation/searching was done quickly and within the time-frame allocated to front-end development. The biggest issues were presented when using the Reportlab library which proved to be more complicated than expected. Moreover, for days with a large number of trades (over 2000 trades) the system requires around 10 seconds before a PDF is generated. This issue is counteracted by the fact that the system is expected to generate the report on its own at the end of the day rather than throughout the day. In conclusion, the implementation for Daily Reports meets the requirements set for the system (Functional req. 5.1D-5.4D) whilst being simple and easy to use.

3.1.6 Access Administrator

Part of the functional requirements for the system was the existence of an administrator page within which the user would be able to add or remove products, companies, stock values, currencies, currency values. The team decided that the Django template for an admin site could be modified to fit these requirements. As such when the user selects the 'Access Administrator' option from the main menu, they are presented with a list available models (Figure 15) for which they can create new entries as shown for products in Figure 16. This functionality is already provided by the Django framework so simply redirecting to the admin page was enough.

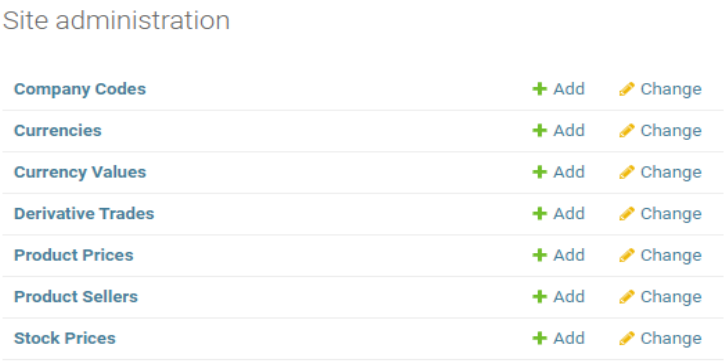

Figure 15: Administrator Model List

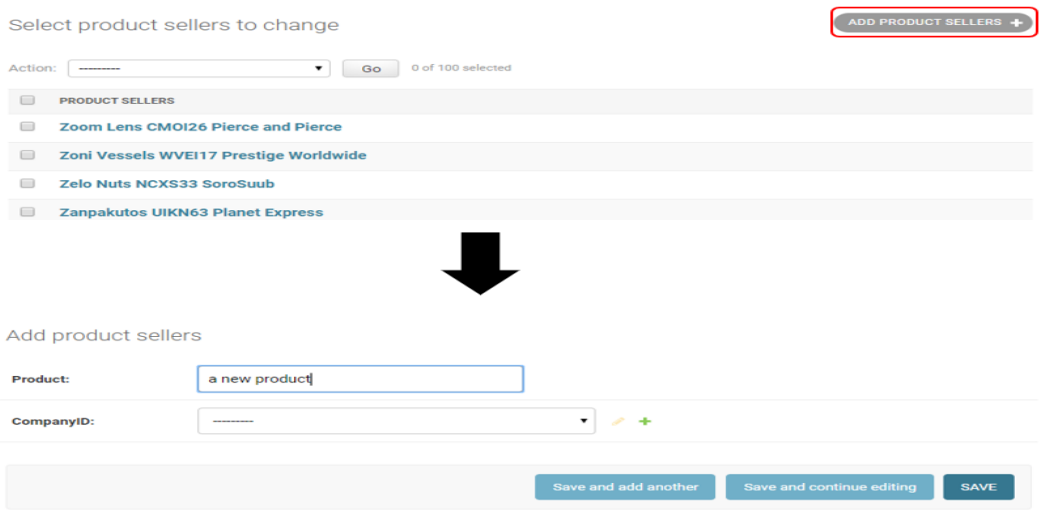

Figure 16: Adding new product example

EVALUATION: The administrator page took the least amount of time to implement as Django's template was already very close to meeting the system requirements with no additional work required from the back-end team. As such all system requirements for the administrator were met (Functional req. 2.3D).

3.1.7 Upload CSV

Although not a system requirement, the 'Upload CSV' feature of the system was implemented to help with the testing of the system's speed when reading a large volume of data as well as provide an alternative way of input to the database, specifically one which did not require interaction with any forms or manual input. When the user clicks the 'Upload CSV' button, they are presented with a form as shown in Figure 17. The form uses an upload field which prompts the user to select a CSV file to upload. When the upload button is clicked the `UploadCSV()` function defined in views.py is called. The file name of the CSV is very important as it is used to determine the model for which new objects are being created. Valid names include companyCodes, productSellers, productPrices, stockPrices, currencies, currencyValues and derivativeTrades. The file is validated to ensure that it is a CSV file before each entry is used to construct the appropriate object. Lastly, the file structure of the CSVs must match that of the provided test data.

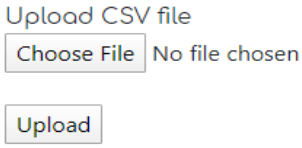

Figure 17: CSV Upload Form

3.2 Back End

The back end's primary role is providing automatic validation and error correction of user input (Functional req. 7) and hence all the relevant functions are placed in the machine_learning.py file. In accordance with the design plan, the user input validation is performed on StrikePrice and Quantity fields of an entered trade. All other fields are assumed to be correct as soon as they pass front-end validation.

3.2.1 Statistical Analysis & Machine Learning

To satisfy the requirement of identifying acceptable data thresholds (Functional req. 7.1D), two separate ordinary least squares (OLS) simple linear models are created each time an input check against all trades of the same Product from the same sellingParty currently stored in `DerivativeTrades` data table is performed, with the dateOfTrade (in the form of days since 01/01/2010) as the explanatory variable and strike price (expressed in USD) and quantity as the response variables respectively. Both models have intercept, slope and regression standard error (RES) coefficients associated with them, which are essential in the area of prediction and hence anomaly detection as described in a further section.

The models as detailed above are generated in the `generate_models()` function using utilities offered by *NumPy* (array and matrix operations), *pandas* [8](dataframe processing) and *SciPy* (scientific inc. statistical computing) libraries. This function takes the selling party (*selling_Party*) and product (*product*) names as well as the date of trade (*trade_date*) as input

and returns 6 coefficients - *pi*, *pg*, *psd*, *qi*, *qg*, *qsd* (intercept, slope and RES for corresponding strikePrice p and quantity q given days since 01/01/2010 d linear models) as follows:

$$\left.\begin{array}{l} selling\_Party \\ product \\ trade\_date \end{array}\right\} \xRightarrow{generate\_models()} \begin{cases} \overbrace{pi}^{intercept}, \overbrace{pg}^{slope}, \overbrace{psd}^{RES} \\ \underbrace{qi}_{intercept}, \underbrace{qg}_{slope}, \underbrace{qsd}_{RES} \end{cases}, \quad where \begin{cases} p = \overbrace{pi + pg * d}^{\hat{p}} + \epsilon_p \quad and \quad \sum_{i=1}^{n} \frac{\hat{\epsilon}_p^2}{n-2} = \sum_{i=1}^{n} \frac{(p-\hat{p})^2}{n-2} = psd^2 \\ q = \underbrace{qi + qg * d}_{\hat{q}} + \epsilon_q \quad and \quad \sum_{i=1}^{n} \frac{\hat{\epsilon}_q^2}{n-2} = \sum_{i=1}^{n} \frac{(q-\hat{q})^2}{n-2} = qsd^2 \end{cases}$$

Firstly the verified historical trade data for the given *selling_Party*, *product* and from 365 days prior to *trade_date* is fetched from the database and converted into a *pandas* dataframe. Then, dateOfTrade and strikePrice columns of the that dataframe are converted into days since 01/01/2010 and values in USD respectively, with the process of the latter requiring a join operation (offered by *pandas* merge() function) with fetched from `CurrencyValues` into another dataframe corresponding currency exchange rates. As the simple linear regression *SciPy* linregress() function takes *NumPy* arrays as input, columns of the dataframe are converted and provided to the function. Thus, intercept and slope coefficients are obtained for both models. What follows is the computation of residual standard devation (RES). Finally all 6 coefficients are returned with the first 3 being associated to strikePrice linear model (and the last 3 to quantity linear model).

EVALUATION: It should be noted that the use of *SciPy* library for computing linear model coefficients was not initially assumed. As stated in the design document the *Scikit* machine learning library was to play that role, but has been replaced due to *SciPy*'s superior compatibility with *NumPy*. What is more, at the beginning of the implementation process the team decided to use the online learning - recursive least squares (RLS) algorithm [9], thus avoiding the computational cost of filtering the whole `DerivativeTrades` database and the $\Omega(n^3)$ (due to matrix multiplication) regression function every time a new trade is entered. The necessary functions have even been implemented (*recursive_least_squares*, *simple_linreg_predict*, *new_s_error_param*, *s_error*) relying heavily on matrix manipulation provided by the *Numpy* library. However, the necessity of storing additional parameters in a new data table due to the recursive nature of the proposed algorithm and untraceability of the obtained results leading to model adjustment inability (removing erroneous trade records) have been the decisive factors in abandoning this approach in favour of the traditional ordinary least squares (OLS) predictors. Finally, such a decision has been facilitated by the fact that, as it has since turned out, the additional computational costs in the scale of the system pose no challenge to modern processors of the server, meaning the system remains responsive regardless of the volume of the data stored (Non-functional req. 2).

3.2.2 Correction Algorithm

The correction algorithm aims to generate potential user intended inputs when the input for either StrikePrice or Quantity is outside of two standard deviations from the expected value. It is implemented in the form of two functions called `error_check()` and `generate_possible_values()` in the machine_learning.py. The `generate_possible_values()` function takes two inputs; a float and a boolean. The float is a representation of the input from the user that the model has deemed likely to be incorrect. The boolean is for the types of corrections that should be attempted, when true the algorithm assumes the input was typed on a num-pad, when false it assumes the input has been typed on a number-line. This is because fat-finger errors will be different across the two inputs. The application always assumes that inputs are typed on a number line.

The algorithm itself generates possible values in the following ways:

- Multiplying the value by 10. This often gives a "good enough" correction when the user has missed a digit within their input. Although more accurate predictions may be generated later in the function.
- Dividing the value by 10. This usually gives a "good enough" correction when the user has double-pressed some digit within their input. Again, more accurate predictions are likely to be generated later in the function.
- Replacing each digit with possible mistyped digits. This involves looking at either the num-pad or number line and selecting all keys which are one position away from the typed digit. For example, 5 on the number line would be replaced by both 4 and 6 and each would be added to the list of possible values. On the num-pad, 5 would be replaced by each of 2, 4, 6, and 8. Doing this for all digits in the number generates multiple possible values.
- Removing duplicate, consecutive digits. In all currently generated values, if two consecutive digits are the same, one of these digits is removed. Since these values already contain the mistyped-digit corrections, removing duplicate digits also corrects values where the user has typed two digits at the same time.
- Replicating duplicate, consecutive digits. Add an extra duplicate digit going from two to three instead of reducing from two to one. Accounts for times when a consecutive key press is missed.

EVALUATION: This variety of error corrections allows the algorithm to correct common errors (Functional req. 7.2D).

3.2.3 Anomaly Handling

When between two and three standard deviations from the expected value, the returned list of values is filtered for values within two standard deviations and these are used to give suggestions to the user. When more than three standard deviations from the expected value, the generated values are filtered for values within two standard deviations, and the value closes to the expected value is used to replace the user input. In the case where no returned value is within two standard deviations, the user is notified that their input is suspected to be incorrect but no correction was found. Once anomalies have been corrected or overridden by the user, they are added to the database. Since each model is generated from the data in the database, the model generated will adapt to include this new data (Functional req. 7.3D).

3.2.4 Database & Models

Development of the database system started with the creation of the user interface and front-end components. The team decided it was important to implement the database near the start of development, as it would make testing the other components much easier given direct data retrieval and insertion operations being enabled by the database. As stated in the Planning and Design document, the team chose SQLite3, the default Django database system. The database schema detailed in Planning and Design was created using Django models which abstract the process of writing SQL directly, resulting in the following models being defined in the cs261_webapp/application/models.py file: `CompanyCodes`, `ProductSellers`, `Currencies`, `ProductPrices`, `StockPrices`, `CurrencyValues`, `DerivativeTrades`. Each table field was implemented using the Django model field equivalent with the only notable exception being the Archived field for derivative trades where instead of using a Bit field, Boolean was used instead. Data retrieval operations where carried out using Django QuerySets on a given database model. This functionality allowed for model objects to be retrieved using appropriate filtering on each model field. Moreover, the use of foreign key relations in the `DerivativeTrades` model allowed for the use of autofields in the front-end component of the system.

After the completion of implementation of the database, the test data provided for the year 2019 were inserted to ensure proper functionality. Data retrieval and insertion requests were handled correctly by the database, however a minor flaw was discovered through some basic testing of the 'Daily Reports ' component of the system. Although the research conducted by the back-end team showed that SQLite3 performs the majority of the operations of the system faster than most alternatives [10], the volume of data stored by the system meant that fetch requests of the entire database or of similar scope could slow the system down significantly. Such requests however where not expected to occur during normal operation of the system but rather only when the database is erased or populated from scratch.

EVALUATION: The implementation of the database system was completed within the nine day time slot allocated to its development and although functional, a small risk of system slow down existed when the database is erased or populated for the first time. Given that no such operation is expected to occur aside from the initial population of the database, the implementation was deemed successful as per the requirements of the system.

4 Testing

Testing for the system consisted black box testing for both integration testing and unit testing(for unit testing: valid and invalid inputs, input syntax and numeric boundaries tests for unit testing) as described in the module slides [11].

4.1 Unit Testing

4.1.1 Create a New Trade

The create a new trade page was tested to ensure that its validation met the specification and that requirement 6C was fulfilled. The following Fields were automatically type validated by Django and so required no further validation: Quantity and Strike Price. The fields Buying Party, Selling Party, Notional Currency and Underlying Currency had drop down menus, and so no validation was needed as there was only a finite number of options, each one acceptable. For testing on the trade ID, check table 1, the product field and requirement 2.3D see table 2 and for the date fields, see table 3

Table 1: Validating the Trade ID Field

Test Data	Expected Outcome	Success/Failure
Trade ID: 12345678	Rejected: Need 16 characters	Success
Trade ID: 12345678876543211	Rejected: Need 16 characters	Success
Trade ID: 1234567887654321	Rejected: Need 8 characters followed by 8 numbers	Success
Trade ID: abcdefgh12345678	Accepted: Follows the required pattern	Success
Trade ID: abcdefg!12345678	Rejected: ! is not an accepted character	Success

Table 2: Validating the Product Field

Test Date	Expected Outcome	Success/Failure
Product: Scope Lens	Accepted: valid product	Success
Product: Scope Len	Rejected: Not a valid product	Success
Product: Scope Lens!	Rejected: Incorrect character, but also invalid product	Success
Product: Scope Lens, Buying Party: KTLA69	Buying Party does not sell that product	Success
Product: Scope Lens, Buying Party: UACN81	Accepted as Buying Party sells the product	Success

Table 3: Validating the Date of Trade and Maturity Date

Test Data	Expected Outcome	Success/Failure
Date: 01/01/2009	Rejected: Before the first accepted Date	Success
Date: 02/01/2019, Maturity Date: 01/01/2019	Rejected: Maturity Date after Date	Success

Table 4 shows the unit tests on Underlying Price and Notional Amount which are auto generated by the system. By allowing the system to generate these values prevented the user inputting incorrect values for these fields, and removed the need to even check for fat finger errors. This meant that once the system was tested to ensure it worked exactly as expected, these fields of the database would never be incorrect.

Table 4: Underlying Price and Notional Amount Testing

Test Data	Expected Output	Success/Failure
Quantity: x , Notional Currency:USD, Price:1	Notional Amount is x (i.e unchanged)	Success
Quantity: x , Notional Currency:USD, Price:y	Notional Amount is x.y	Success
Quantity:x, Notional Currency:GBP, Price:y	Notional Amount: x.y.(rate of conversion)	Success
Price: x, Underlying Currency: USD	Underlying Price: x	Success
Price:x, Underlying Currency: GDP	Underlying Price: x.(rate of conversion)	Success

4.1.2 Editing and Deleting Trades

When editing a trade, the user is presented with the option to edit the value of any field (except the underlying price and notional amount). The same validation for these fields applies as before, and so the system was tested to ensure it met the same requirements as when a user added a new trade.

The search facilities on the edit page allows a user to search for a trade, as specified in requirement 3.2D and 4.2D. To search bar at the top allowed searching by attributes such as trade ID, product and buying and selling parties. These are tested in table 5. Filtering by dates and editing trades based on their date and the time that has passed since their creation is tested in table 6

Table 5: Testing the Search a Trade Feature

Test Date	Expected Outcome	Success/Failure
Trade ID: GCJHDCUA50652718	Single correct trade should appear	Success
Product: Blue Shells	List of trades, each with product equalling blue shells	Success
Buying Party: OKYS51	List of trades, each with buying party equalling OKYS51	Success
Selling Party: BBJG05	List of trades, each with buying party equalling BBJG05	Success
Any Incorrect Value	The system should handle this exception	Success
Filtered by date and then search	The system applies both filters to the resulting list	Success

Table 6: Testing the Date Filtering Options

Test Data	Expected Results	Success/Failure
Filter by a certain time	Only the trades appear occurring within that time limit	Success
Edit a trade happening 24 hours and a minute ago	Changes not committed	Success
Edit a trade happening 23 hours and 59 minutes ago	Changes committed	Success
Edit a trade created a minute ago	Changes committed	Success
Edit field and make an error	Correct validation response	Success
Field edited, and then daily report opened	Changes should be updated	Success
Try to edit trade older than day, then create daily report	Report remained consistent	Success
Delete a trade happening 24 hours and a minute ago	—Not Deleted	Success
Delete a trade happening 23 hours and 59 minutes ago	Deleted	Success
Delete a trade created a minute ago	Deleted	Success
Delete trade and then open daily report	The trade has been deleted	Success
Try to delete trade older than a day and then open the daily report	Report remains consistent	Success

4.1.3 Producing the Daily Report

The daily report is generated when a user wants to view all the trades occurring from a specific day. The testing conducted on both this feature and the searching daily reports can be found in the table 7, testing requirements 5C, 5.2D and 5.4D.

Table 7: Testing Daily Report Generation and Report Searching

Test Data	Expected Outcome	Success/Failure
User compiles the daily report for a date within 2010-2019	The report is generated, with only trades from that day	Success
User compiles report from before 2010	Error message	Success
User compiles report from after 2019	If data exists, produce PDF. Otherwise create a blank one	Success
Trades and edited/deleted and then compiled	Report matches the changes	Success
Searching today's report	The most up to date report is returned	Success
Report searched from 10/12/2019	The report is loaded	Success
Report searched from 10/12/1999	Error flagged as date out of range	Success

4.1.4 Database

The database was tested consistently by outputting the results of queries, and iterating through the total output set, to check if the correct values have been stored and retrieved in the correct fields. The testing conducted can be found in table 8

Table 8: Testing functionality of the database

Test Data	Expected Outcome	Success/Failure
Request of all CompanyCodes objects	QuerySet of all CompanyCodes objects returned to calling function	Success
Request of all ProductSellers objects	QuerySet of all ProductSellers objects returned to calling function	Success
Request of all Currencies objects	QuerySet of all Currencies objects returned to calling function	Success
Request of all ProductPrices objects	QuerySet of all ProductPrices objects returned to calling function	Success
Request of all StockPrices objects	QuerySet of all StockPrices objects returned to calling function	Success
Request of all CurrencyValues objects	QuerySet of all CurrencyValues objects returned to calling function	Success
Request of all DerivativeTrades objects	QuerySet of all DerivativeTrades objects returned to calling function	Success
Model save request for CompanyCodes object	CompanyCodes object saved	Success
Model save request for ProductSellers object	ProductSellers object saved	Success
Model save request for Currencies object	Currencies object saved	Success
Model save request for ProductPrices object	ProductPrices object saved	Success
Model save request for StockPrices object	StockPrices object saved	Success
Model save request for CurrencyValues object	CurrencyValues object saved	Success
Model save request for DerivativeTrades object	DerivativeTrades object saved	Success

4.2 Integration Testing

Once the front end was complete, and the machine learning model had been finished, integration testing had to be performed to ensure that the two subsystems could work in tandem [12], shown in table 9. Now that the back end had been tested and certifiably returned correct errors, the system had to be tested to ensure that they were displayed to the user correctly and errors were autonomously corrected.

Table 9: Integration Testing

Test Data	Expected Outcome	Success/Failure
New trade made using trades form with valid data	Check for field & model errors and save trade	Success
New trade made using trades form with Quantity within 2 SD	Display error with suggestions within 2 SD	Success
New trade made using trades form with Quantity within 3 SD	Display error and replace user input	Success
New trade made using trades form with Quantity within 3 SD and no historical correlation	Display error	Success
New trade made using trades form with StrikePrice within 2 SD	Display error with suggestions within 2 SD	Success
New trade made using trades form with StrikePrice within 3 SD	Display error and replace user input	Success
New trade made using trades form with StrikePrice within 3 SD and no historical correlation	Display error	Success
User overrides StrikePrice correction	Accept user feedback and add to correction model	Success
Edit trade using trades form with valid data	Check for field & model errors and save trade	Success

4.3 Model Testing

The machine learning model's trade data anomaly detection results have been investigated to assess its performance and evaluate the success of the project. The sample data used for this purpose consisted of all trades of *Ultra Balls* sold by the entity *SAAM06* in 2019. The team has established that the sample is representative for all other trades (of different products from different sellers in any time frame) and hence can be used to come up to conclusions regarding the whole system component.

4.3.1 Assumptions and Effectiveness

Since a linear regression model has been chosen for anomaly detection, a number of assumptions has to be satisfied to ensure its valid predictable behavior:

1. The errors are uncorrelated
2. The errors have equal variance
3. The errors have zero mean

Additionally, another assumption can be checked

4. The errors are normally distributed

as this provides numerous extra mathematical properties of the model.

A visual indication of whether these assumptions are true or not is acquired by examining a plot of the residuals $\hat{\epsilon}$ against the fitted values $\hat{y}$. Scatter and residual plots in Figures 18, 19, 20 provide answers to the question of assumptions 1.-3. validity, whereas histograms of residuals and Q-Q plots allow to evaluate the truthness of assumption 4.

Moreover, such a graphical representation has provided insight into data considered historical and real as well as, more importantly, the model's behaviour on it by identifying trade entries found to be suspicious, which should hence be corrected or confirmed by the user, and allowed the team to gauge the the solution's performance.

4.3.2 StrikePrice model

By examining plots contained in Figure 18 the following conclusions can be reached:

- Assumption 1 is not satisfied (errors seem to be correlated), which is a natural effect of (easily observable) periodical fluctuations in the strike price.
- Assumptions 2 and 3 (errors with equal variance and zero mean) are satisfied.
- The additional assumption 4 is not satisfied (as the distribution of the residuals is bimodal), although the residuals are close to being distributed normally.

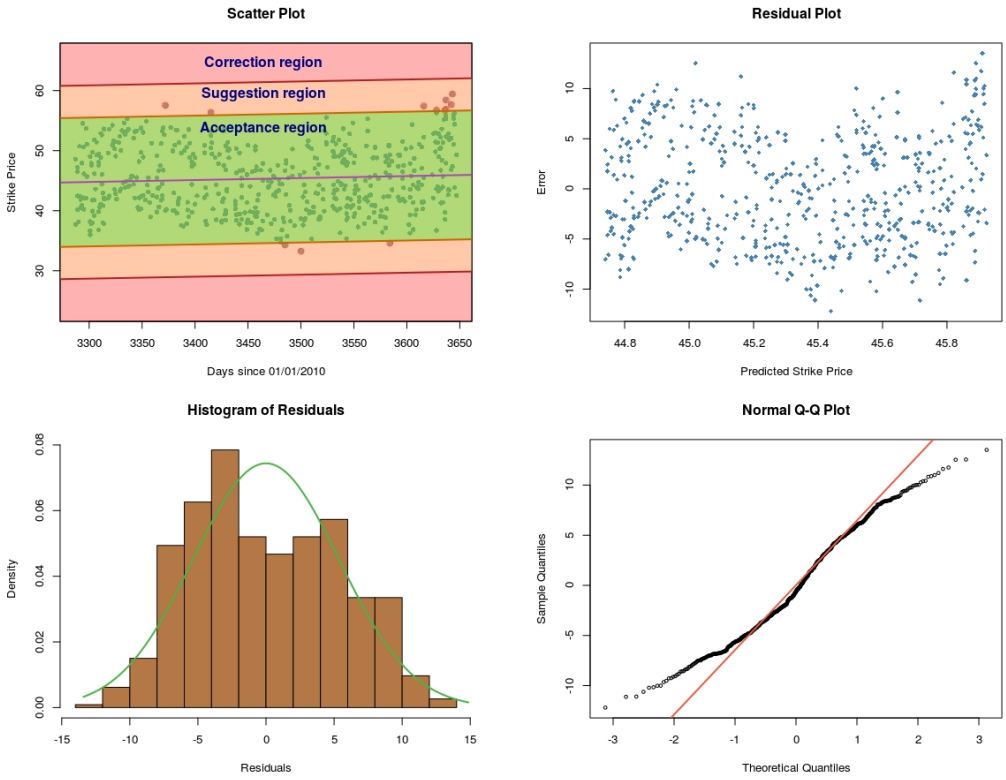

Figure 18: Strike Price model evaluation

Unfortunately, the behaviour of the model cannot be considered fully correct due to error correlation resulting from data periodical trends. This issue could be potentially overcome by identifying strike price monotonicity periods and creating the model based on data from only one such period at a time. This would be done instead of the team's generally reasonable decision to use trade data from 365 days prior an entered trade. Overall, however, the model should work as desired in the majority of cases as the errors have mean zero and have equal variance. Nevertheless, the bias-variance trade-off apparent in any machine learning model could in this case be alleviated by a regularization method[13].

Moreover, due to an almost normal distribution of errors, the model functioning offers an extremely valuable asset our correction algorithm depends on. Namely, it reflects the 68-95-99.7 rule [14], which states that under the normality assumption 5% (data points further than 2 standard deviations from zero) of entered trade values would be considered suspicious with 0.3% (data points further than 3 standard deviations from zero) of all entries automatically corrected. As can be seen in plots above, this property is held by the model relatively well, with a small percentage of entries falling into the *Suggestion region*. Thus, the identification of unusual extreme values is ensured.

4.3.3 Quantity model

By examining plots contained in Figure 19 the following conclusions can be reached:

- Assumptions 1-3 (uncorrelated errors with equal variance and zero mean) are satisfied.
- The additional assumption 4 is not satisfied (A vast majority of trade quantities are below 10000, but some are reaching up to 90000 - the error distribution shifted by the expected value reflects this).

In this case, since all the basic assumptions (1-3) are satisfied the behaviour of the model is predictable and can be considered correct. Due to assumption 4 clear failure, however, the model is equipped with 2 obvious imperfections:

Firstly, a significant number of correct entries falls into *Suggestion region* or even *Correction region*, meaning they they are considered suspicious by our solution. Similarly, due to an immense value of RES (unbiased error standard deviation), trades of irregularly small quantities would be far from being spotted and handled appropriately.

A possible method to tackle these issues would be applying the logarithm function to quantity values before computing the model coefficients. The effects of this approach are captured by the plots in Figure 20:

- Assumptions 1-3 (uncorrelated errors with equal variance and zero mean) remain satisfied.
- The additional assumption 4 is still not satisfied, but errors are distributed visibly more evenly.

Unsurprisingly, this solves the issues of not spotting small quantities and incorrectly marking big ones. It comes at cost though, as now all trade entries fall within the *Acceptable region*. While such behaviour could well be desired in real life (after all the studied data is correct), in reality it does not constitute a perfect alternative given our solution: No percentage of the entered trades would deemed suspicious, requiring the quantities to be gigantic this time in order to be investigated.

This just shows that no model can be perfect, and the one chosen by the team is indeed doing its job properly. It shall also be noted that its purpose is not fitting the training data with immaculate accuracy by reverse-engineering it, but rather being able to handle any reasonable data provided to it - as it is.

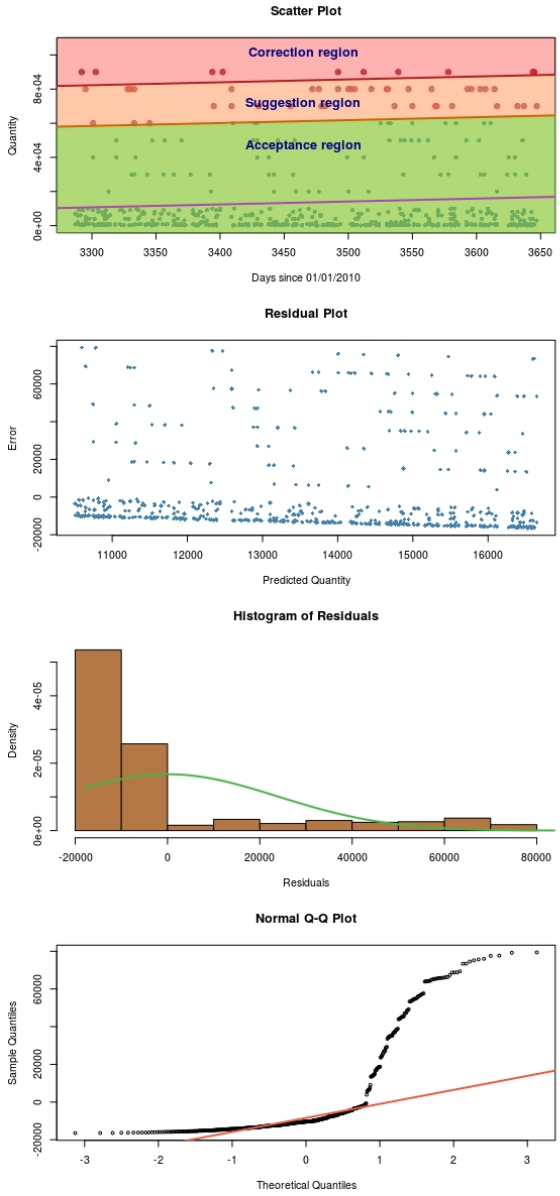

Figure 19: Quality model evaluation

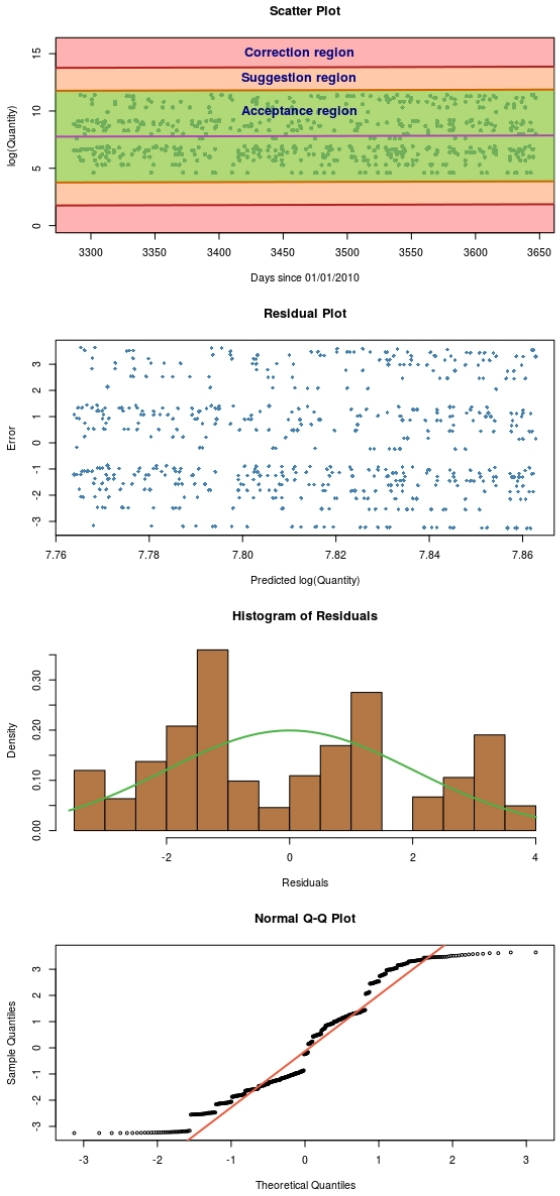

Figure 20: Log(Quality) model evaluation

4.4 Acceptance Testing

There was no way of demonstrating the project to the clients in order to gain feedback. As such, the team showed it to the personal tutor assigned to the group, to hear his suggested improvements. An example would be the original design for the edit a trade page, which he advised could be improved to reduce complexity of the UI, and to make cleaner and more professional. It would also lead to a higher complexity of web design, with the edit a trade facility being located on a new page as oppose to reusing the new trade form.

Taking on board his advice, the front end developers came up with a slicker web form, with increased functionality, that not only met the requirements set, but also exceeded them, providing such additional functionality as viewing a history of changes made to any single form.

However, tutor meetings were only scheduled every other week, and so to ensure that the project was constantly going down the correct path, the team met weekly, and held discussions on how the website looked and acted, and what changes needed to be made to ensure it aligns closer to user requirements. Trivial changes were made weekly; changing fonts, colours, styling and CSS. When discussing larger scope issues, front and back end teams would meet individually, and it was at one of these that the front end developers designed the alterations for the edit a trade page, in light of the tutor's advice.

Therefore, despite not having a client for the team to demonstrate their developing prototype too, by all members critiquing the work and meeting weekly to discuss progress and improvements, this simulated user acceptance testing.

5 Project Management

5.1 Team Roles

Josh handled the role of project manager, meaning he was to ensure the team was kept up to date on the overall development and were meeting deadlines. Stylianos handled the role of business analyst, where he was to keep the team focus on the direction of development. Two teams were formed to delegate the work between the front end and the back end. Stylianos, Josh and Folabi were set to focus on the front end, and Tim, Maciej and Lewis were set to focus on the back end.

5.1.1 Front End

The front end consisted of creating the website, styling it and creating the following features; adding a new trade, editing and deleting an old trade, generating daily reports and searching through them. As such, the group delegated the work for front end between the three front end developers. The editing and deleting a trade functionality was given to Folabi. Adding a new trade and creating the main menu was given to Josh. Generating and searching through daily reports was given to Stylianos. Overall, the front end development was a success. Strict deadlines were set over the entire project, as well as also amongst front end development as well, but every deadline was not only met but exceeded, and as such, a working prototype for the site was completed by as soon as the end of week 6. This is down to good project management (assigning equal work to each individual) and a good work ethic from individual members, to complete their own work and then help out on outstanding tasks from other sections.

5.1.2 Back End

The back end workload was split primarily into three parts: creating the database, writing input validation functions for the Quantity and StrikePrice fields, and designing the statistical analysis and correction component. The work was split accordingly amongst the three back end developers. The database component was given to Lewis. The research and implementation of the statistical analysis aspect was given to Maciej. The correction and validation component was given to Tim along with the connection to the database.

The back end was a success for the group, due to the recognition of the individual skill sets of each member being used efficiently. For example, Maciej's background in Data Science made him the best team member for identifying and justifying a machine learning algorithm, furthermore Tim's experience coding in Python worked well to help implement this algorithm into the solution. Dynamic shifting of team roles was also valuable, as Stylianos (A member of the front end development) helped design the back end database schema due to his experience with Django models. This compromise was able to happen due to how far ahead the development was with the front end, allowing a better parallel development. Despite the back end being difficult to prototype due to its extremely linear implementation, each team member produced their section for the back end, and the final days before the team-set deadline was spent connecting these sections to function together, and also to function with the front end: that of which was successful.

6 Project Evaluation

6.1 Code

The project utilised a web based front end, and a back end written in python, constituting of a machine learning program that learned and developed off of years of statistical data. The Django framework was a brilliant choice of framework, as it abstracted away details of implementation, allowing easier website creation and minimal excessive coding. An example would be the ease by which URLs could be created, then view functions to handle these and render web pages. Such functionality would be incredibly difficult if written without Django, and would have taken far more time to set up and initialise than the framework requires. This meant that because most of the heavy work was automated, creating the web forms and web pages was simplified greatly, reducing the quantity of code written. It also meant the system could utilise tested libraries, reducing the scope of errors in the project, and making it easier to debug and fix when things went wrong.

The system also utilised the functionality of the Django admin page, where it was embedded into the edit and delete web form. Using HTML templates already provided by Django meant that as oppose to creating all pages, the system would override those supplied within the framework, reducing the amount of coding necessary. This not only saved time, but it provided the system with a lot of additional functionality, such as the history of changes for a given trade, which would have been difficult to implement from scratch. The Django admin also had consistent styling, giving the site the professionalism that individually generated web pages might not guarantee. It also provided automatic type validation (such as only allowing numerical characters to be entered into a integer field), reducing the amount of validation necessary to perform, ensuring the front end team could focus on what was important, such as generating values for notional amount and underlying price.

6.2 Sprint Cycles

The team employed the development strategy of sprint cycles for front end development, as it lent itself well to the style of coding. There was a number of different functionality needing implementation, such as creating a new trade, editing trades, deleting trades, generating daily reports and searching them. As the front end could be decomposed into discrete sub components, by having weekly sprint cycles in which the focus on each week was a different component, meant that coding was completed quickly, and entire subsystems were created before the next was even started. This meant that all front end members were aware on how each system was constructed, reducing the bus factor of the project. It also meant that any problems or issues that would arise during the cycle could be handled by the entire team as oppose to a single programmer running into difficulty. This style of programming contributes heavily to the speed at which the front end was completed, and how every deadline was met. It allowed for changes, such as when the edit page was altered from the initial design, to one incorporating and extending upon the Django admin functionality that was provided.

6.3 Management

The project manager of the group was Joshua, and the BA was Stylianos. When the group met each week, the PM delegated the work needed for the following week, and set deadlines that had been decided by the entire group. This meant that every member of the group was happy with the time scale set for various sections of the projects, and any doubts and questions could be raised in front of the entire group. The team then split into their front and back end groups. A communal Whatsapp group chat was created, so that the PM and the team could raise questions and communicate throughout the duration of the project, as well as arrange meetings and discuss ideas. Each sub team also had their own stream on a discord discussion, where front or back end specific details could be talked about only within the relevant team members. The effect of the above management meant that every team member felt included, was up to date on the latest changes on the project, and had the facilities to ask questions or raise queries and have the entire project group available to remedy it. While there was a dedicated PM who communicated with the tutor and set deadlines, the running of the project was also a group effort, and it avoided the situation where team members felt ostracised and communication between all branches of the team was limited at best.

6.4 Work Distributions

The entire team met up and agreed upon the distribution of the work, ensuring that it was not only fair, but every team member was aware of their own responsibilities. The reports were split into sections, and then these were delegated to individual members, who all contributed their work to a shared Overleaf page. The front end work distribution was initially by section, but as the project progressed, it was apparent that sprint cycles, where each subsystem is specifically focused on, was a more effective work distribution, each team member therefore had to ensure that they contributed evenly to each sub component. Back end delegated work slightly differently, with assigning specific roles to each person.

7 Conclusion

7.1 Functional Requirements Success Analysis

Requirement 1: The system provides the user with a menu, consisting of five options; create a trade, edit/delete a trade, produce a daily report and search through daily reports. There is also functionality to add new companies and trades to the database.
Requirement 2: The user is able to create a new derivative trade. The requirement states that the user should be able to enter a value for each entry in the form, which were met, except for notional amount and underlying price which are calculated for the client automatically to remove the possibility of errors.
Requirement 3: The user is able to edit an existing trade. The user can search for these trades through the search bar at the top of the page, and can edit any one of the editable fields (again except for notional amount and underlying price which are generated again for them off of the new inputted values). The way that trades are displayed ensures that no trades after

twenty four hours from creation are displayed, as from that point onwards they are considered archived.
Requirement 4: Using the same page as for requirement 3, the user can search through the list of trades created within the last twenty four hours, and delete any of these. They can search for the trade based on any of the fields, all of which are listed in requirement 4.2D
Requirement 5: The user is able to create daily reports. The system formats these to be user friendly, displaying all trades in ascending order of time on the day selected. All information is stored, including the product, buying and selling parties, quantity and notional amount, exactly as specified in requirement 5C. The user is also able to search by date and find the daily report for an archived date.
Requirement 6: The user input is validated when creating a trade. TradeID is checked to ensure it meets the string structure required, dateOfTrade is validated to ensure it fits within the time frame and that the maturity date proceeds it and buying and selling parties have drop down menu UI's in place to ensure that only valid entries are submitted into the DB.
Requirement 7: The back end of the system utilises machine learning to learn from previous years of trade data and determine if there are any errors in the data entered. It can determine when it believes fat finger errors have occurred, and automatically corrects the value, notifying the user of the change, as well as the suggested correction. It also notifies the user when it believes an error to have occurred but when it does not have a suggestion.

7.2 Project Summary

The task was to create a prototype system that was to facilitate the ability to create new derivatives trade, checking for errors in all inputs, store these trades, and then create daily reports for any date, with the ability to search these. The team created a team hierarchy, with Josh being the project manager and Stylianos being the business analyst, and met weekly to delegate work and suggest improvements.
The front end team created the website, utilising the Django web framework because of its high level of documentation and ease of use. They created pages to facilitate adding a new trade, editing and deleting trades and creating and searching through daily reports. The back end team researched machine learning models and implemented one in python, which took data from the UI, entered by the user, and checked the ten years of previous data to determine whether the input was correct and within lines of what would be expected, using standard deviation to determine how many steps away from the ideal an entered value was.

This project had a deadline for ten weeks, with other deadlines set intermediaries throughout that period for deliverables. The team created a requirement document to outline the requirements of the task and what would be expected of the solution, and then created a design document outlining the approach the team would be taking to meet the requirements set.

Once these two documents were due, the team began programming in earnest, implementing the designs and writing the final report, testing to ensure it worked as expected and then evaluating the performance. Every requirement was hit, and within the time frame. The system works as expected, with additional functionality than originally planned, and although the solution deviates from some of the initial designs, any changes are well documented and with good reason.

Overall, the project was a success. The team delivered the product, as intended, within the deadline. It completes exactly what it is meant to do, and to a high level of code, documentation and design. Each team member had work delegated to them, and was responsible for their own portion of the overall project. Effective time management, structure and work delegation was key to the success of the project, and meant that the deliverables, and then the final product, were all submitted to a high standard.

References

[1] "Scipy documentation." [cited 03 March 2020]. Available from: https://docs.scipy.org/doc/scipy/reference/.

[2] "Numpy documentation." [cited 03 March 2020]. Available from: https://numpy.org/devdocs/reference/index.html#reference.

[3] "Django models." [cited 06 March 2020]. Available from: https://docs.djangoproject.com/en/3.0/topics/db/models/.

[4] "Django template language." [cited 06 March 2020]. Available from: https://docs.djangoproject.com/en/3.0/ref/ templates/language/.

[5] "Django class based views." [cited 06 March 2020]. Available from: https://docs.djangoproject.com/en/3.0/topics/class-based-views/.

[6] "Difflib for python 3." [cited 05 March 2020]. Available from: https://docs.python.org/3/library/difflib.html.

[7] "Reportlab pdf library version 3.5.36," Feb 05, 2020. [cited 05 March 2020]. Available from: https://www.reportlab.com/docs/reportlab-userguide.pdf.

[8] "Pandas library documentation version 1.0.1," Feb 05, 2020. [cited 04 March 2020]. Available from: https://pandas.pydata.org/pandas-docs/stable/.

[9] S. Boyd and S. Lall, "Recursive estimation." [cited 06 March 2020]. Available from: http://ee263.stanford.edu/lectures/recursive.pdf.

[10] "Database speed comparison." [cited 03 March 2020]. Available from: https://www.sqlite.org/speed.html.

[11] J. Archbold, "Topic 7: Software testing." [cited 08 March 2020]. Available from: https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs261/l14-1.pdf.

[12] "Integration testing." [cited 05 March 2020]. Available from: http://softwaretestingfundamentals.com/integration-testing/.

[13] J. E. Gonzalez, "Linear regression and the bias variance tradeoff." [cited 06 March 2020]. Available from: http://people.eecs.berkeley.edu/ jegonzal/assets/slides/linear_regression.pdf.

[14] M. Evans and H. Moshonov, "The empirical (68-95-99.7) rule." [cited 06 March 2020]. Available from: http://fisher.utstat.toronto.edu/ hadas/STA220/Lecture%20notes/week3.pdf.