MarketGeek

Goldsmiths
UNIVERSITY OF LONDON

Final Deliverable for [redacted]

Report on the process of carrying out the software project and the non-software outputs.

Authors: [redacted]

[redacted]

Table of Contents

Abstract 4

Outcomes 4

Introduction 4

Planning and Research 5

- Resource and time allocation 5
  - Frontend 5
  - Back-end 7
    - Computational resources, CUDA 8
  - Breakdown of the team 8
  - Project Management Tools 9
  - Dataset 10
- Topic Modelling 10
  - Latent Dirichlet Allocation 10
  - BERTopic 11
  - BERTopic vs LDA 14
- Sentiment Analysis 16
- Clustering 17

Design and Development 19

- Colour blindnesses 19
- User testing 19
- Design - first iteration 20
- Overall - second iteration 25
- Heuristic evaluation 28
- Initial Design 30
- Production prototype, first iteration 32
- Secondary Design 32
- Components of the software 34
- Presentation layer 34
- Structure and Implementation 36
- Browser Support 37
  - Supported 37
  - Recommended 38

The API server 39
JSON Web Token 40
Sign up and sign in 41
Modules 45
Endpoints 46
MongoDB 47
Jupyter Notebook 48
TailwindCSS 50
React.js 52
Search page and functionality 55
Response Attributes 57

Evaluation 58
Back-end 58
Conceptual issues 58
Technical issues 63
Front-end 69
Technical issues 69

Conclusion and summary 73
Back-end 73
Front-end 73
Overall 74

References 75

Appendix **80**
Work history 81

Abstract

The purpose of this project is to create a functional tool, in which users can gain insight for the analysis of many assets (for example, SNP500, gold, crude oil) based on historical and daily news articles. This provides a representation of how historical events influenced the market in the past in order to better understand current daily trends. The tool itself can be used for anyone interested in financial research, whether it is for someone desiring to monitor price changes, or to get an idea of market movement to support their own investment decisions.

Based on the user testing results provided in the section “User Testing”, this team has successfully managed to develop the basis of the product, with the intention to add more features and functionality in future work.

Outcomes

As for overall goals for this stage of the project, this team expects users to be able to create or login to an account on a range of devices that allow for internet connectivity to then be taken to their own personal dashboard. In the dashboard a user can expect to find different features including lists of assets that they are interested in, and the corresponding price and change of price. The ability to check news, especially anything influential that has caused changes in any holdings. A settings page that allows the user to make basic changes to their account, such as password, name, email and a photo. This team expects users to be able to do all of the aforementioned tasks whilst also not having issues navigating the site.

Introduction

The aims for this stage of the project are to successfully develop the web application. The team is divided into two front and three back-end developers, with one full stack developer. The back-end team will focus on the functionality of the algorithms that make up the financial analysis tool while the front-end team will focus on presenting the information in a clear and concise way that easily shows the different trends of an asset based on feedback from testing. Throughout the development of the project this team will be referring back to the initial concept report in which the premise and modelling prototypes were laid out.

This report contains an extensive body of research, followed by user testing, which includes heuristic evaluation, and a brief overview of technologies that were used in the development along with their pros and cons. And finally, an in-depth analysis of the work done that includes technical issues (in both back-end and front-end) as well as conceptual issues.

Planning and Research

Resource and time allocation

Frontend

The first two weeks were allocated to get familiar with the different libraries and frameworks we proposed to use, such as setting up a new React project or prototyping with TailwindCSS using an online playground called **Tailwind Play** (**Figure 1 and 2**). This allowed everyone to prototype the proposed high fidelity designs and do some additional interactive user testing (as shown in User testing in Section “Design and Development”) on these designs.

In addition, this team researched the most common industrial ways to collaborate on the same codebase and set up the different Git repositories for the frontend, full stack and back-end team on GitHub. The team also created an internal document with important information about the project, such as resources this team used to learn new technology, user credentials for the public Linux Server and other short tutorials. (“Software Project – Getting Started”)

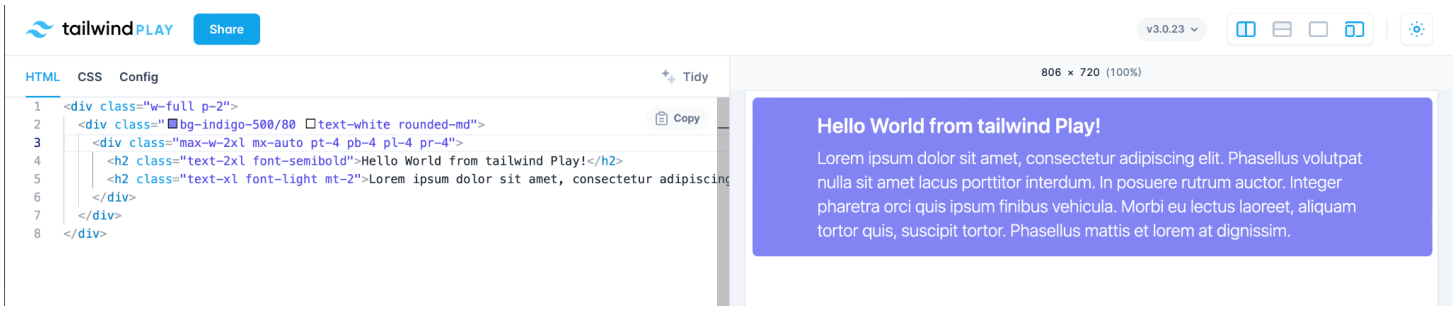


Figure: 1: *“Hello world” tailwind play example (above)*
Figure: 2: *Design and development tailwind (below)*

Front-end time and progression log

Computer Project 2 - Marketgeek Project "Front-end" team					
			Project progression log		
Tasks	**Start Date**	**Duration**	**Complete**	**Incomplete**	**Participants**
MarketGeek project initial design planning	[redacted]	3 hours	Overall app layout	Code implementation	[redacted]
Initial libraries installation, setting up the MarketGeek web server, setting up programming environment (React, Node.js, Tailwind CSS)	[redacted]	16 hours	Set up a new Node.js project React learning process HTTP Server implementation Connect to the MongoDB NPM module API endpoints	Project implementation in code	[redacted]
SignIn and SignUp pages and account page implementation	[redacted]	10 hours	Completed functionality of SignUp form and SignIn page, created Jira project version control.	data preload, Search page and Search bar, Sign in modal	[redacted]
Web User Authentication	[redacted]	5 hours	Sign in to the existing account. MarketGeek Server API.	data preload, Search page and Search bar	[redacted]
A search box component	[redacted]	7 hours	The component implements a search through stocks, posts and other things on MarketGeek. This component is TextField used in the navigation bar on the home page when a user is logged in.	data preload on dashboard, data preload on search page	[redacted]
Page for Terms of use and Privacy Policy	[redacted]	4 hours		data preload on dashboard, data preload on search page	[redacted]
Finalising dashboard	[redacted]	8 hours	Widgets on the dashboard Design tweaks, Positive news sentiments, Sentiments scalable representation, Top news	data preload on search page	[redacted]
Finalising search page	[redacted]	7 hours	Data preload on the search page, External API		[redacted]

Table 1: *Progress log for front-end team*

Back-end

The back-end team had to make sure they were comfortable with certain concepts and skills. As such, two weeks in the beginning of the term were allocated specifically for that. The back-end team focused on: Python, machine learning, natural language processing, and data analysis techniques overall. Both teams knew that this approach would take a significant amount of time, but it was necessary to successfully reach the desired goals. After this learning process, both teams had at least two meetings per week to issue code reviews and work collaboratively on the project. Outside of this time, individual members continued to work on portions of the project that were allocated to them using the kanban through Jira.

Weekly meetings for the full group including front and back end took place. In these meetings the team discussed as a whole the status and progress of the project. Detailing specific updates that had taken place during the week. These meetings were used to troubleshoot and discuss the interaction between the front end web application and the back-end functionality.

Back-end time and progression log

Computer Project 2 - Marketgeek Project "Back-end" team

			Project progression log		
Tasks	**Start Date**	**Duration**	**Complete**	**Incomplete**	**Participants**
Searching for a financial news dataset	[redacted]	1 week	Reuters and Bloomberg datasets found	Could not locate contemporary dataset	[redacted]
Researching, learning and becoming familiar with the following topics and skills: Python, machine learning, natural language processing, and data analysis.	[redacted]	2 weeks	All members became familiar with the required skills	Lacking comprehensive understanding of some more complex concepts including neural networks	[redacted]
Cleaning and sorting the financial news dataset.	[redacted]	1 week	Dataset successfully cleaned and pre-processed		[redacted]
Natural language processing: BERTopic, LDA, FinBERT.	[redacted]	4 weeks	Performed topic modelling and sentiment analysis on a given dataset		[redacted]
Obtain current news dataset	[redacted]	1 day	Successfully obtained current news from NewsAPI		[redacted]

Find price trends in several financial assets	[redacted]	3 days	Successfully obtained starts and ends of price trends for given financial assets	Some price trends were inconsistent: upwards trends had downward movement and vice versa.	[redacted]
Find co-occurrences and export JSON	[redacted]	2 weeks	Found co-occurrences for five random news articles	Could not connect the back-end functionality to the front-end web application	[redacted]

Table 2: *Progress log for backend team*

Computational resources, CUDA

During the project it became apparent that the computational methods the team were using were very extensive and resource consuming. After experimenting with the components associated with the Microsoft Azure API server it became apparent that the team needed a system with more computing power. As a result, the team researched and implemented the CUDA toolkit ("CUDA Toolkit", n.d.). As part of our agile development scheme, we were able to incorporate this after development had begun. CUDA toolkit is an NVIDIA application process that allows GPU (graphics processing unit) enhanced processing. Defined as general purpose computing on a GPU, CUDA works as the interface that gives access to the instruction set of the GPU, allowing for parallel computing to take place. The CPU runs as a single thread, completing the sequential part of the process while the GPU takes on the demanding workload. This would allow a much more time efficient approach for the computationally heavy and numerous tasks the back-end team had to complete.

Breakdown of the team

The team is made up of 6 members, all of which are named at the top of the document. Upon the creation of this team it was decided to split the group into smaller sub groups. Dividing into the sub groups would allow for better management of time and allow people within the groups to focus on more specific areas that they are interested in, with the 2 groups being front-end and back-end with one person taking the role of full stack. The sub teams mainly work within their own domain on tasks related to the project, having meetings multiple times a week, along with frequent meetings with the other sub team to work on the overall functionality of the application.

Project Management Tools

Jira is a web application that offers a project management tool that provides the ability to assign tasks to specific people, groups or the whole team whilst maintaining a structured base. The ability to give each task a description, set specific tags or a reference to the GitHub repository turned out to be very helpful during the planning stage and development process of the project. The tool has become widely used by agile development teams to track bugs, stories, epics, and other tasks. (“What Is Jira?”, n.d.)

The MarketGeek project development team integrated Jira into its project version control process and linked Jira with Git.

The time and progression log (see **Table 1**) represents the front-end teamwork progress on the MarketGeek project according to Jira project version control updates. Column “Tasks” describes all main tasks the front-end team set for this project and “Start Date” as well as “Duration” represent the actual time frame the team spent on resolving a given task. “Complete” describes completed tasks and “Incomplete” the tasks which are still in progress.

The last column “Participants” represents the names of front-end team members allocated to a given task.
(**why, reference, version control, alternatives, advantages, why is it the best system, centralised/decentralised**)

GitHub is a tool that allows us to use Git in a simple user interface. Git is a technology that helped this team to separate code bases into different repositories and sync changes that are made to files and folders between multiple contributors. The teams made use of branches within the repositories to improve the version workflow between working different issues, branches were deleted after another person within the same team verified and merged the code into the master branch of the repository. This all happened in a short code review process where both parties went through the lines of code that have changed, what things could be improved and giving general feedback on how the issue was completed.

At the end of the project and during building a public version of the web application, we made use of the release feature to distribute a released working version on GitHub using its version system.

For this application, it is important to make sure that every developer is able to make changes to the codebases without losing existing parts of the code made by someone else. This can be important when two team member are making changes to the same file or the same line(s) and want to prevent overwriting changes from each other. (“A beginner's guide to using Git when working with a team for the first time” 2020)

The version control tool allows the team to track back changes, the commits will keep track of the changes we make during a push or merge.

Dataset

For this application it is vital to have a dataset containing news articles from a trustworthy source. That is why the free Bloomberg dataset with daily news from 2006 to 2013 was a reasonable choice. Bloomberg is a well-known news platform, which provides reliable and unbiased news [1]. Another reason would be that there are not a lot of quality news datasets, which possess relevant content and are of the desired timeframe, especially the free ones.

Whilst looking for a dataset this team ran into a few issues. First of which was the trouble of finding a dataset that was in a presentable form. Many of the datasets that were found were in wildly complicated formations that would require time to unscramble, for which there was not enough time. Another issue was the timeframes of the dataset's, all of which were not up to date with current years, which meant that it would be necessary to extrapolate data to today's date. It was decided to settle on the issue with the dates as an up to date dataset that was free could not be found.

Topic Modelling

Topic modelling is statistical modelling where an algorithm discovers abstract topics from some collection of documents. There are a lot of algorithms to tackle this problem, but only a few were tried by this team (Li 2018). The main purpose of using topic modelling in this implementation was to generate a number of topics per document without resorting to keyword matching, which would require a lot more time to carry out.
A lot of research has been done to determine the best algorithm for assigning topics to every news article for the sake of better representation. Below a reader can find a brief explanation of the explored options along with their comparisons.

Latent Dirichlet Allocation

The first option this team explored was Latent Dirichlet Allocation (LDA).
LDA is an algorithm that was and is still used widely across multiple industries. It shares some similarities to a dimensionality reduction technique called Principal Component Analysis, because they are both algorithms of matrix factorisation (Whye and Guide 2021).

First, there are two assumptions this algorithm makes: documents are a mixture of topics and topics are a mixture of words. In statistical language, the documents are known as distributions of topics and topics as distribution of words (Blei, Ng, and Jordan 2003).

Before it comes to applying the algorithm, the data has to be cleaned. It implies that there should be no punctuation, numbers, stop words and all the words have to be either stemmed or lemmatised. This is done because all these elements are just noise (Dua 2021). After that, the algorithm constructs a document-word matrix, in which every row is a document and every column is a word.

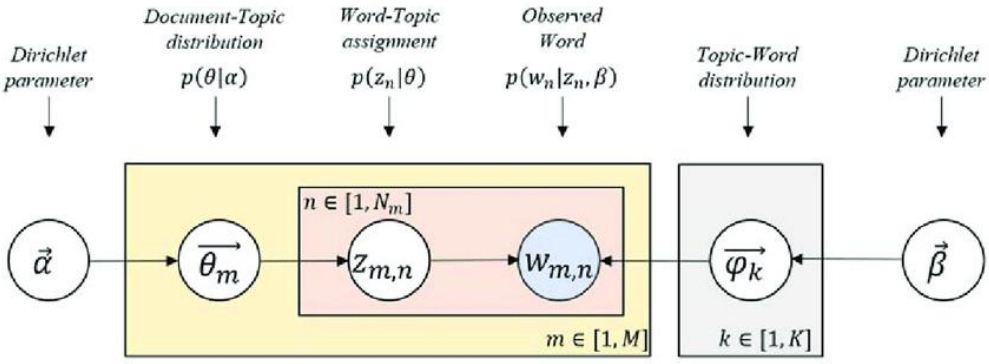


Figure 3. *LDA scheme. (Lee, Junseok et al. 2018)*

Then, LDA performs matrix factorization and splits the document-word matrix into document-term matrix and topic-word matrix. Document-term matrix contains possible topics that a document can contain, while topic-word matrix contains words that those topics can contain.

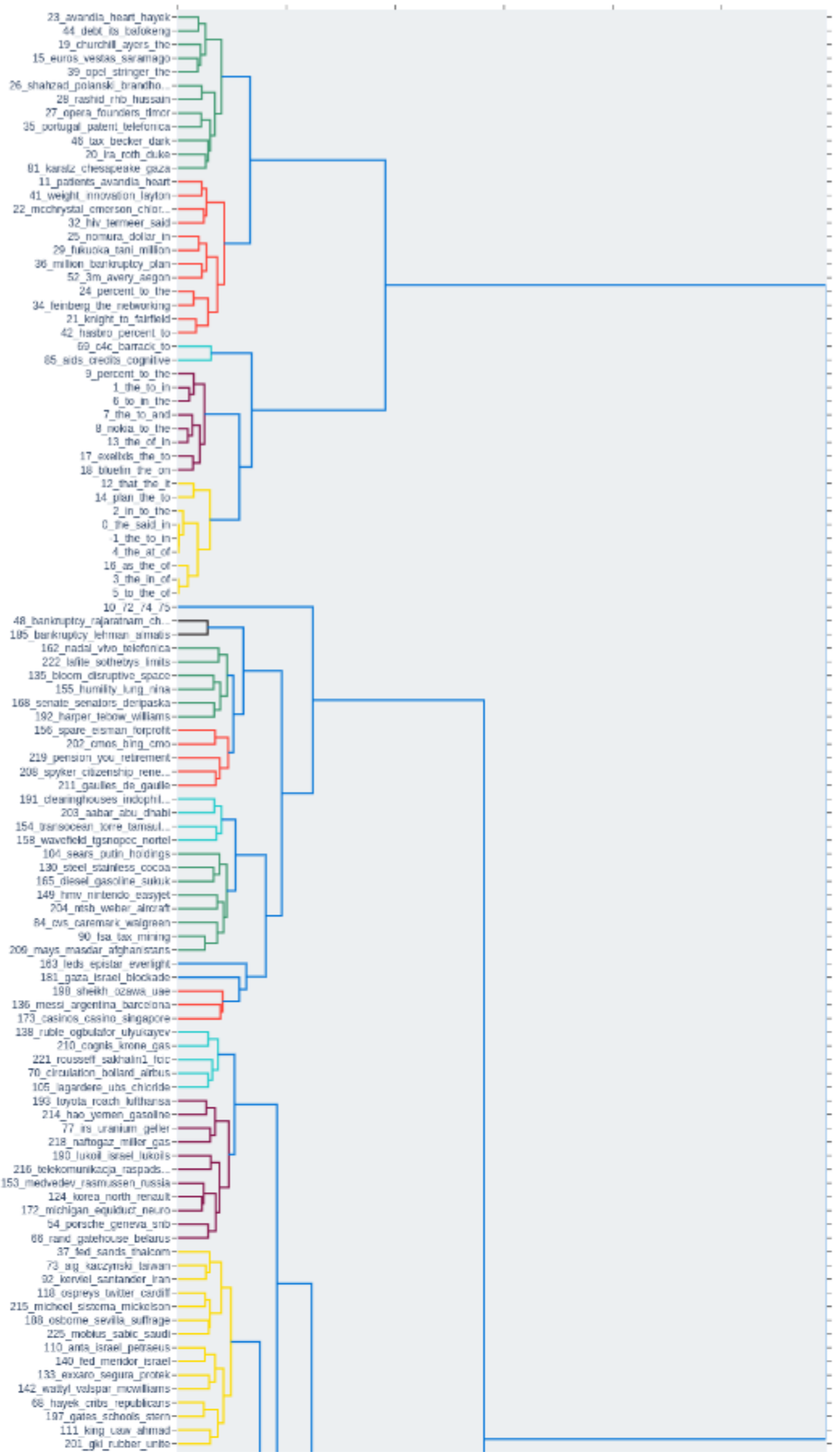
According to LDA, every word is associated with a latent topic, which is stated by Z, whereas topic-word distribution is represented by Θ.

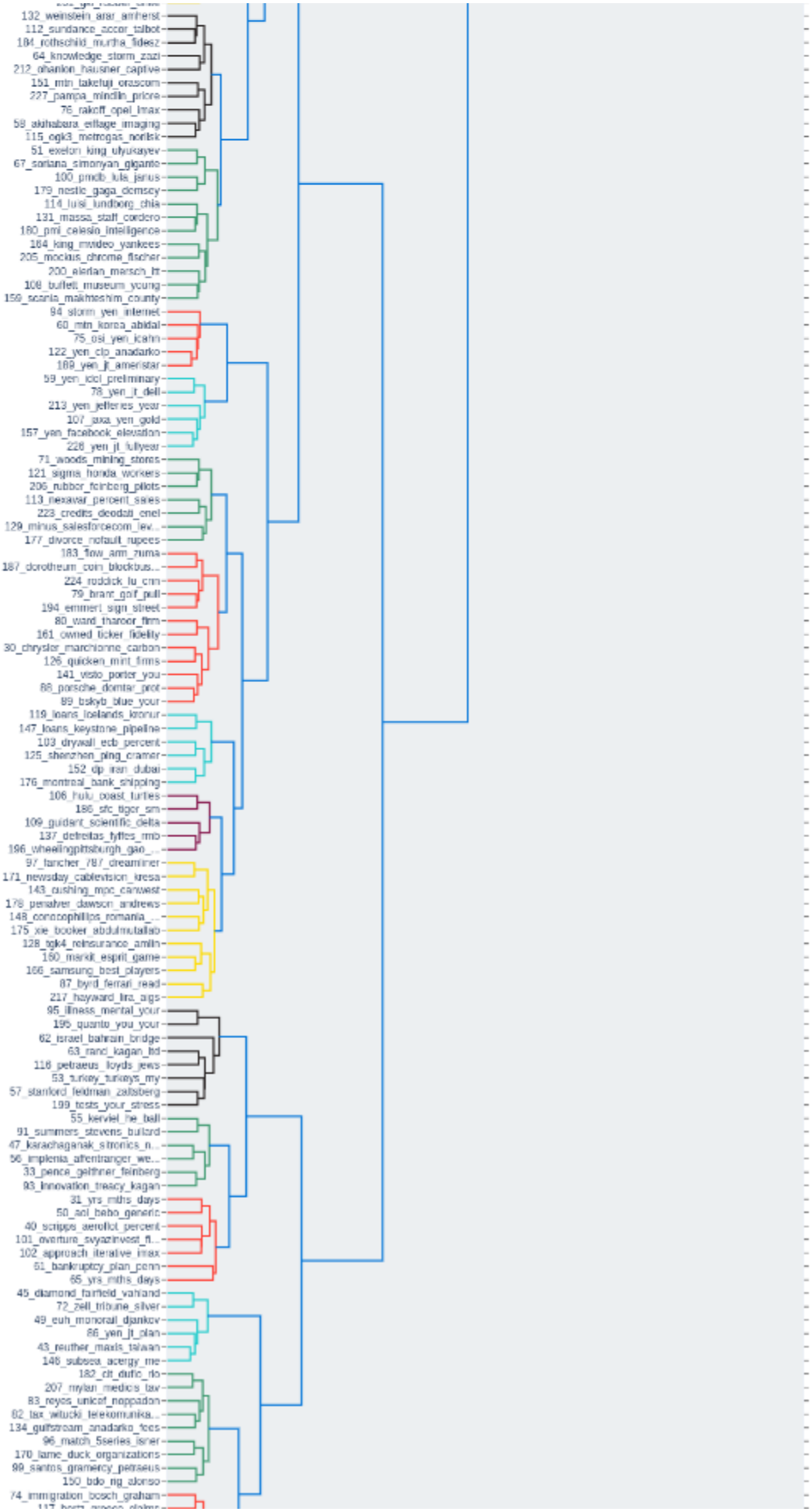
BERTopic

BERTopic is the second topic modelling technique that the team tested. This technique uses sentence transformers to produce embeddings and HDBSCAN to divide it into clusters. Once the embeddings are divided into clusters, BERTopic uses c-TF-IDF to determine relevant words that represent those clusters (Grootendorst, n.d.).

c-TF-IDF stands for class-based term frequency-inverse document frequency. TF-IDF is a technique used to measure the relevance of a word in the context of a document or collection of documents. c-TF-IDF is a class based variation. This procedure uses scikit-learn's TfidfTransformer as a base.

Each class of topics is converted to a single document. The frequency of certain words are extracted from each class and divided by the total number of words. BERTopic can be used for guided, supervised and dynamic topic modelling.


Hierarchical Clustering
23_avandia_heart_hayek
44_debt_its_bidokeng
19_churchill_ayers_the
15_euros_vestas_saramago
39_opcl_stringer_the
26_shahzad_polanski_brandho...
28_rashid_rhb_hussain
27_opera_founders_dmcr
35_portugal_patent_telefonica
46_tax_becker_dark
20_ira_roth_duke
81_karatz_chesapeake_gaza
11_patients_avandia_heart
41_weight_innovation_layton
22_mcchrystal_emerson_chlor...
32_hiv_temeer_said
25_nomura_dollar_in
29_fukuoka_tani_million
36_million_bankruptcy_plan
52_3m_avery_aegon
24_percent_to_the
34_feinberg_the_networking
21_knight_to_fairfield
42_hasbro_percent_to
69_c4c_barrack_to
65_aids_credits_cognitive
9_percent_to_the
1_the_to_in
6_to_in_the
7_the_to_and
8_nokia_to_the
13_the_of_in
17_exelids_the_to
18_bluefin_the_on
12_that_the_it
14_plan_the_to
2_in_to_the
0_the_said_in
-1_the_to_in
4_the_at_of
16_as_the_of
3_the_in_of
5_to_the_of
10_72_74_75
48_bankruptcy_rajaratnam_ch...
185_bankruptcy_lehman_armalis
162_nadal_vivo_telefonica
222_lafite_sothebys_limits
135_bloom_disruptive_space
155_humility_lung_nina
168_senate_senators_deripaska
192_harper_tebow_williams
156_spare_esman_forprofit
202_cmos_bing_cmo
219_pension_you_retirement
208_spyker_citizenship_rene...
211_gaulles_de_gaulle
191_clearinghouses_indophil...
203_aabar_abu_dhabi
154_transocean_torre_tamsul...
158_wavefield_tgsnopec_nortel
104_sears_putin_holdings
130_steel_stainless_cocoa
165_diesel_gasoline_sukuk
149_hmv_nintendo_easyjet
204_ntsh_weber_aircraft
84_cvs_caremark_walgreen
90_fsa_tax_mining
209_mays_masdar_afghanistans
183_leds_epistar_everlight
181_gaza_israel_blockade
190_sheikh_ozawa_uae
136_messi_argentina_barcelona
173_casinos_casino_singapore
138_ruble_ogbulafor_ulyukayev
210_cognis_krone_gas
221_rousseff_sakhalin1_for
70_circulation_holland_airbus
105_lagardere_uhs_chloride
193_toyota_roach_lufthansa
214_hao_yemen_gasoline
77_its_uranium_geller
218_naftogaz_miller_gas
180_lukoil_israel_lukoils
216_telekomunikacja_raspads...
153_medvedev_rasmussen_russia
124_korea_north_renault
172_michigan_equidux_neuro
54_porsche_geneva_snb
66_rand_gatehouse_belarus
37_fed_sands_thalcom
73_sig_kaczynski_taiwan
92_kerviel_santander_iran
118_ospreys_twitter_cardiff
215_michael_sissoma_mickelson
188_osborne_sevilla_suffrage
225_mobius_sabic_saudi
110_anta_israel_petraeus
140_fed_mendor_israel
133_aixaro_segura_protek
142_wadlyl_volspar_mcwilliams
68_hayek_cribs_republicans
197_gates_schools_stern
111_king_uaw_ahmad
201_gkl_rubber_unite


132_weinstein_arar_amherst
112_sundance_accor_talbot
184_rothschild_murtha_fidesz
64_knowledge_storm_zuzi
212_ohanlon_hausner_captive
151_mtn_takehuji_orascom
227_pampa_mindlin_prices
76_rakoff_opel_lmax
58_akihabara_effage_imaging
115_ogk3_metrogas_nordisk
51_exelon_king_ulyukayev
67_soriana_simonyan_gigante
100_pmdb_luis_janus
179_nestle_gaga_domscy
114_lusi_lundberg_chile
131_massa_staff_cordero
180_pmi_celesio_intelligence
184_king_nvideo_yankees
205_mockus_chrome_fischer
200_alerian_mersch_ftr
108_buffett_museum_young
159_scania_makmeshim_county
94_storm_yen_internet
60_mtn_korea_abidal
75_dsl_yen_icahn
122_yen_clo_anadarko
189_yen_jt_ameristar
59_yen_icici_preliminary
78_yen_jt_dell
213_yen_jefferies_year
107_jaxa_yen_gold
157_yen_facebook_elevation
228_yen_jt_fullyear
71_woods_mining_stores
121_sigma_honda_workers
206_rubbor_feinberg_pilots
113_nesawar_percent_sales
223_credits_deodati_enel
129_minus_salesforcecom_lev...
177_divorce_notault_rupees
183_low_arm_zuma
187_dorotheum_coin_blockbus...
224_ruddick_lu_cnn
79_brant_golf_pull
194_emmert_sign_street
80_ward_tharoor_firm
161_owned_ticker_fidelity
30_chrysler_marchionne_carbon
126_quicken_mint_firms
141_visto_porter_you
88_porsche_domtar_prot
89_bskyb_blue_your
119_loans_icelands_kronur
147_loans_keystone_pipeline
103_drywall_ecb_percent
125_shenzhen_ping_cramer
152_dp_iran_dubai
176_montreal_bank_shipping
106_hulu_coast_turtles
186_sfc_tiger_sm
109_guidant_scientific_delta
137_detroitas_tyffes_rmb
196_wheelingpittsburgh_gao_...
97_fancher_787_dreamliner
171_newsday_cablevision_iresa
143_cushing_mpc_carwash
178_penalver_dawson_andrews
148_conocophillips_romania_...
175_xie_booker_abdulmutallab
128_tgk4_reinsurance_amlin
180_merkit_esprit_game
166_samsung_best_players
87_byrd_ferrari_read
217_hayward_lira_algs
95_illness_mental_your
195_quanto_you_your
62_israel_bahrain_bridge
63_ranci_kagan_ltd
116_petraeus_lloyds_jews
53_turkey_turkeys_my
57_stanford_feldman_zalisberg
199_tests_your_stress
55_kerviel_he_bail
91_summers_stevens_bullard
47_karachaganak_sitronics_n...
56_implenia_affentranger_we...
33_pence_gelfriner_feinberg
93_innovation_treacy_kagan
31_yrs_mths_days
50_aol_baho_generic
40_scripps_aeroflot_percent
101_aventure_svyazinvest_ft...
102_approach_iterative_lmax
61_bankruptcy_plan_penn
65_yrs_mths_days
45_diamond_fairfield_vahland
72_zell_tribune_silver
49_euh_monorail_djankov
86_yen_jt_plan
43_reuther_maxis_taiwan
148_subsea_acergy_me
182_clt_dutic_rio
207_mylan_medics_tax
83_reyos_unicef_noppadon
82_tax_wiucki_telekomunika...
134_gulfstream_anadarko_fees
96_march_5series_isner
170_lame_duck_organizations
99_santos_gramercy_perseus
150_bdo_ng_alonso
74_immigration_bosch_graham

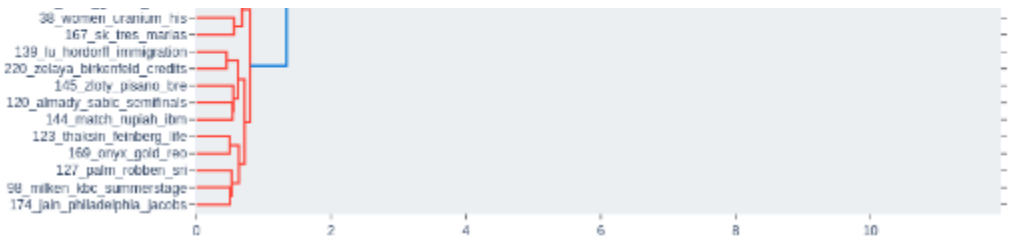


Figure 4: *BERTopic hierarchy visualised on a random 10000 samples.*

BERTopic vs LDA

As outlined above, LDA assumes the given text will be a mixture of topics and certain words will be associated with their respective topics. LDA works by giving a probability vector for each body of text that belongs to a topic.

In contrast, BERTopic uses first HDBSCAN clustering followed by c-TF-IDF (refer to the above section) which relies on the frequency of words in the body of text. The part which provides embeddings (BERT) is pre-trained.

The team tested both approaches on the given dataset. The topics provided by both algorithms were consistent, however, the BERTopic library provided more accessible functionality, including several visualisations of the data in question, which was useful for the purpose of testing and documentation.

One of the main things the team had to consider was the difficulty and time investment of learning the concepts and implementation of the different methods for topic modelling. As a result, the team researched and tested both methods. However, the documentation and existing sources of information for BERTopic were presented in a more accessible manner, and as a result easier for the team to learn. In addition, BERTopic also has the ability to detect outliers in the text. This is a valuable feature as it filters any data that is too generic. What was fundamentally important to the project was that BERTopic allows us to get the cluster IDs that correspond to each topic. The cluster IDs are convenient for producing the co-occurrences.

Lastly, and most importantly, BERTopic has shown to outperform LDA based on the metrics called topic coherence and topic diversity. Topic coherence (Rosner et al. 2012) has a range [-1,1] and it has shown to imitate human judgement reasonably well, while topic diversity (Dieng, Ruiz, and Blei 2019) has a range [0,1] where 0 indicates redundancy in the output consisting of topics and 1 indicates an exclusive variety of those topics.

	20 NewsGroups		BBC News		Trump	
	TC	TD	TC	TD	TC	TD
LDA	.058	.749	.014	.577	-.011	.502
NMF	.089	.663	.012	.549	.009	.379
T2V-*MPNET*	.068	.718	-.027	.540	-.213	.698
T2V-*Doc2Vec*	.192	.823	.171	.792	-.169	.658
CTM	.096	.886	.094	.819	.009	.855
BERTopic-*MPNET*	.166	.851	.167	.794	.066	.663

Figure 5: *Results of the models with topic numbers ranging from 10 to 50 with a step of ten, and averaging across 3 runs for each step, with a total of 15 separate runs per score (Grootendorst 2022)*

.

As can be seen in Figure …, BERTopic largely outperforms LDA. However, it has not shown better results than CTM (Correlated Topic Models) for topic diversity. This team considered trying the CTM model, which is an extension of LDA (Blei and Lafferty) in future work.

Sentiment Analysis

Sentiment analysis is the process of taking text data and interpreting its overall emotional component. The outcome is a distribution of either positive, negative or neutral sentiment (Gupta, n.d.). It is widely used in many industries, for instance, Intel, Twitter and IBM are implementing it in their workflow to determine employee concerns. As quoted by Justine Brown “Such tools utilise analytics to sort through huge collections of data and help human-resource managers figure out how workers feel about the company” (Brown 2015).

The sentiment of news articles are essential for determining the trend direction of a potential asset as a consequence of world events. Sentiment of financial news shapes the behaviour of a large proportion of potential investors and traders, both of which are key potential stakeholders.

When it came to sentiment analysis, this team decided to use a pre-trained transformer model called FinBert (“ProsusAI/finBERT: Financial Sentiment Analysis with BERT”, n.d.). FinBert is a model available on GitHub, and is based on transformers, which is a powerful neural architecture heavily used in language processing. Essentially, BERT is a stack of transformers, but unlike other language modelling techniques that try to solve a language modelling task, which is predicting the next word, BERT masks 15% of tokens in the corpus that are chosen randomly and then tries to predict them. Authors of FinBERT took a standard BERT model and pre-trained it on a large corpus of financial text, taken from datasets such as TCR-2 Financial, Financial PhraseBank, and FiQa statement.

Finbert outperformed other techniques designed for these tasks, such as regular LSTMs and ULMfit, with the final test accuracy score being 97%. There are, however, situations where FinBert might provide incorrect results (Araci 2019). For instance, the model draws conclusions without the words indicative of direction. The authors provided the following example: “Pre-tax loss totaled euro 0.3 million, compared to a loss of euro 2.2 million in the first quarter of 2005.”. In the sentence above, the true value is positive, however the predicted value is negative. Overall, 73% of misclassifications were between positive and neutral, since it is hard to infer if the sentence is just a mere observation, or a positive outlook.

Clustering

One of the fundamental technologies included in this project is a clustering algorithm called Hierarchical Density-based spatial clustering of applications with noise, or HDBSCAN (Malzer and Baum 2021) . This algorithm was used implicitly in BERTopic. Clustering algorithms are a family of unsupervised learning algorithms that are designed to divide the data points into groups based on their similar traits (Al-jabery and Wunsch II 2022).

Clustering algorithms are divided into 4 categories: flat Centroid based, hierarchical centroid based, flat density based, hierarchical density based.

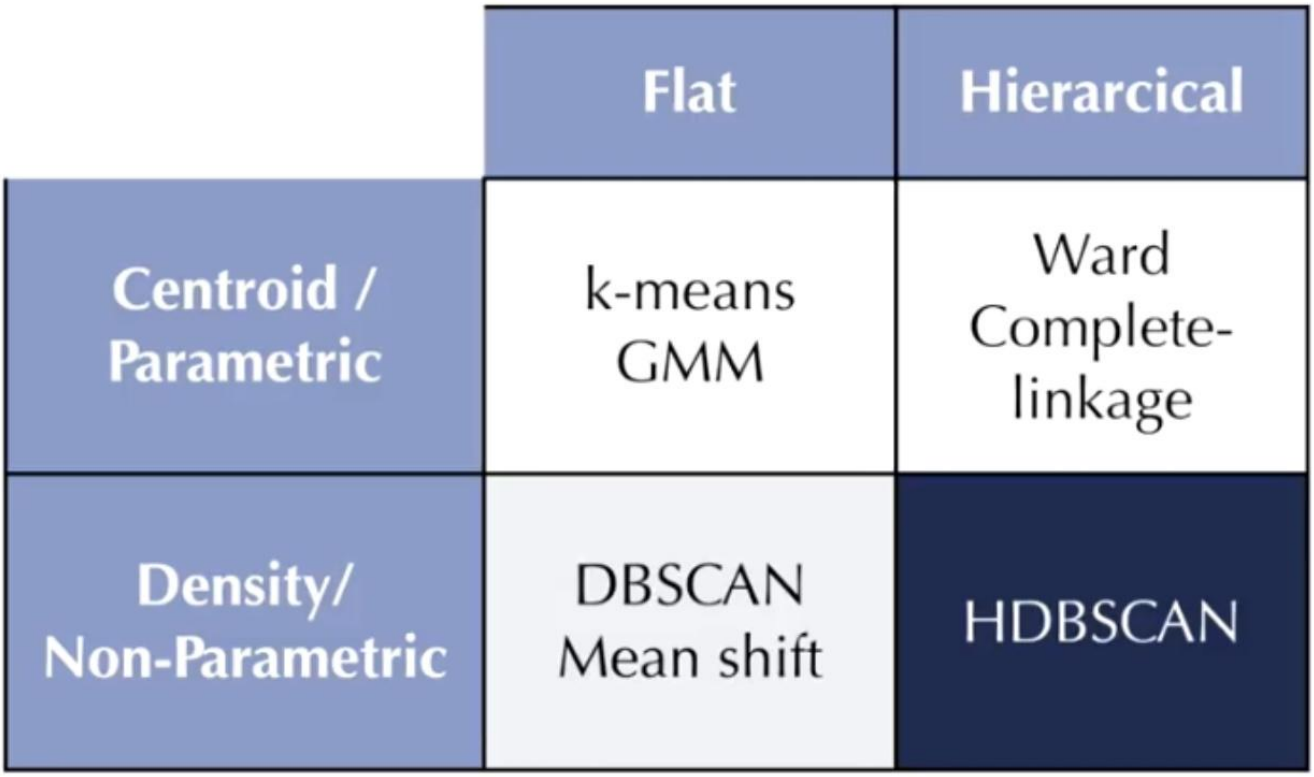


Figure 6: *Types of clustering algorithms.* ("HDBSCAN, Fast Density Based Clustering, the How and the Why - John Healy" 2019)

Flat Centroid based approaches work well for low dimensional datasets, however, they tend to have assumptions about cluster shape and size, which hinders flexibility for most of the real world tasks. It is also a common problem for this set of algorithms to suffer from data outliers, since they do not consider separating true data from the noise (Stanford edu 2009). Additionally, choosing the parameters for these algorithms also poses a problem. When it comes to flat hierarchical based algorithms, they eliminate the challenge of parameter selection,

but they still suffer from cluster shape assumptions, which is again rare in nature, even though theoretically appealing. When considering flat density based algorithms, problems like cluster shape assumption are solved, but again, as with flat centroid based algorithms, finding the right parameter values remains problematic (Campello et al. 2020).

However, hierarchical density based approaches like HDBSCAN solve all of the aforementioned issues while giving efficient, robust performance.

Efficiency of this algorithm comes from the fact that it utilises spatial indexing trees for computing the nearest neighbours, which is the essence of this approach.

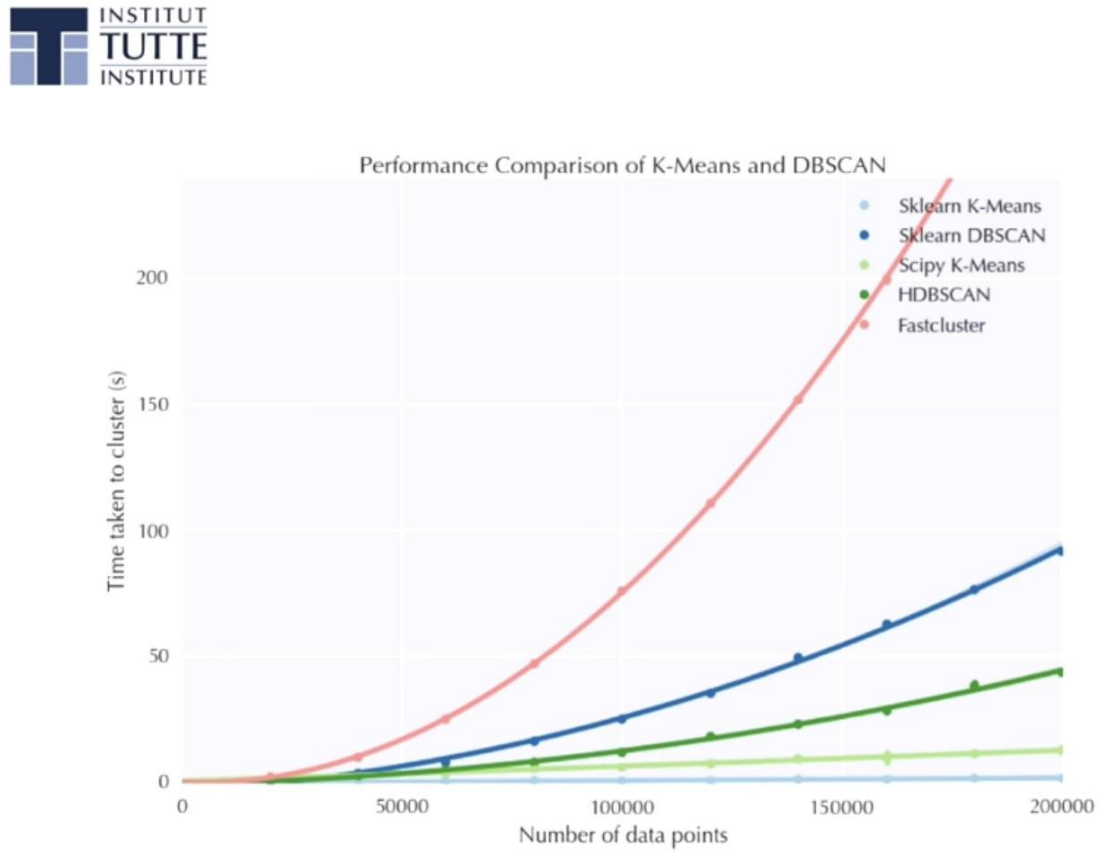


Figure 7: Efficiencies of clustering algorithms.

Notice, time complexity of DBSCAN is less than that of HDBSCAN, even though the latter is equivalent to running the former for every epsilon value. This was achieved by using computational tricks, such as spatial indexing trees. These characteristics are the exact reason why the author of BERTopic used this approach in his work.

Design and Development

Colour blindnesses

Due to the different colour blindness in the world, we made sure to test the web application under Red/Green, Green/Red and Blue/Yellow simulations. The tests were done using the Accessibility features on macOS under the maximum available intensity and on different pages within the application.

Deuteranopia has shown many similarities to this design without the colour filters, for example: the sign out button at the top right corner (see Figure 8) still gives the user a flagging signal when signing out. This team experienced the same effect on the other colour filters as well. All other information towards colour blind consideration can be found in our pre-consideration in the previous review document ([redacted] 2022).

Figure 8: *Sign-out button*

User testing

From what can be seen in the data from our initial high fidelity testing ([redacted] 2022), our final high fidelity model had successful scores from all respective questions. This positive feedback allowed us to move on for the front end development team to take the mockups and create the real designs for the application.

The first production prototype testing was done after a design was implemented for a landing page for the web application, and later on, further parts of the dashboard were tested. The first of the iterations was mainly focused on design and consisted of 4 testers with differing levels of familiarity with the type of application. The testers were either instructed to create an account or were given a pre-made account, this allowed us to see a different movement that testers took on the pages. The team received positive feedback for our initial designs with some criticism that allowed us to make changes to the next round of prototypes that also included more functionality testing alongside final design testing.

Design - first iteration

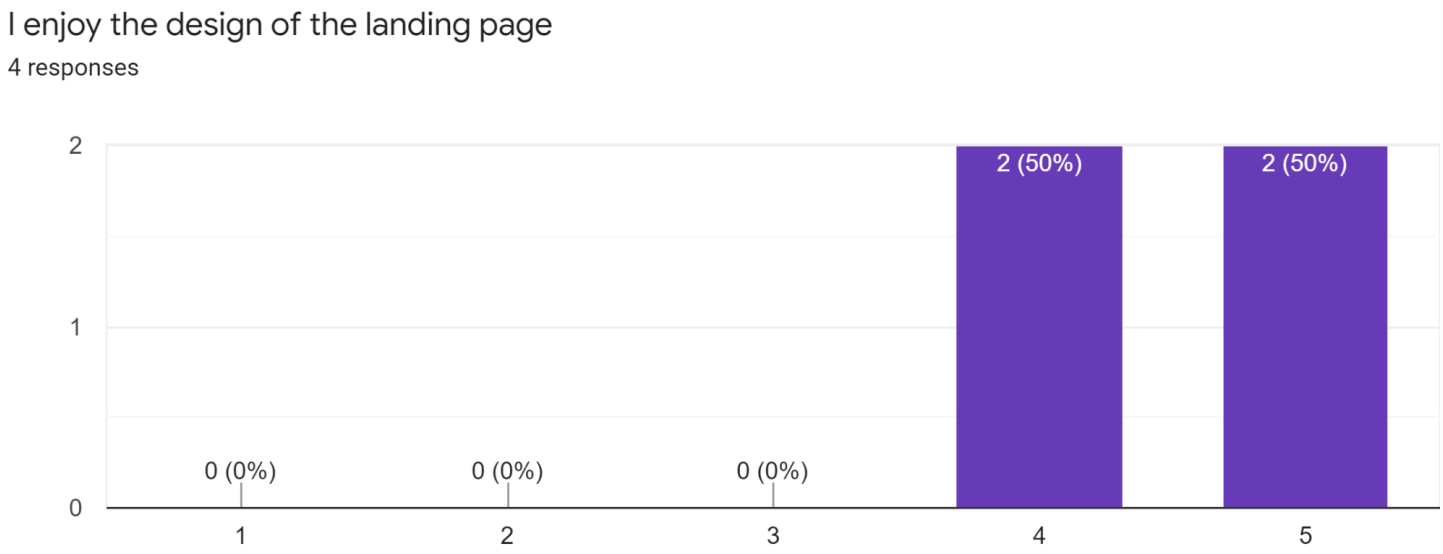


Figure 9: *User Testing Iteration 1 = "I enjoy the design of the landing page"*

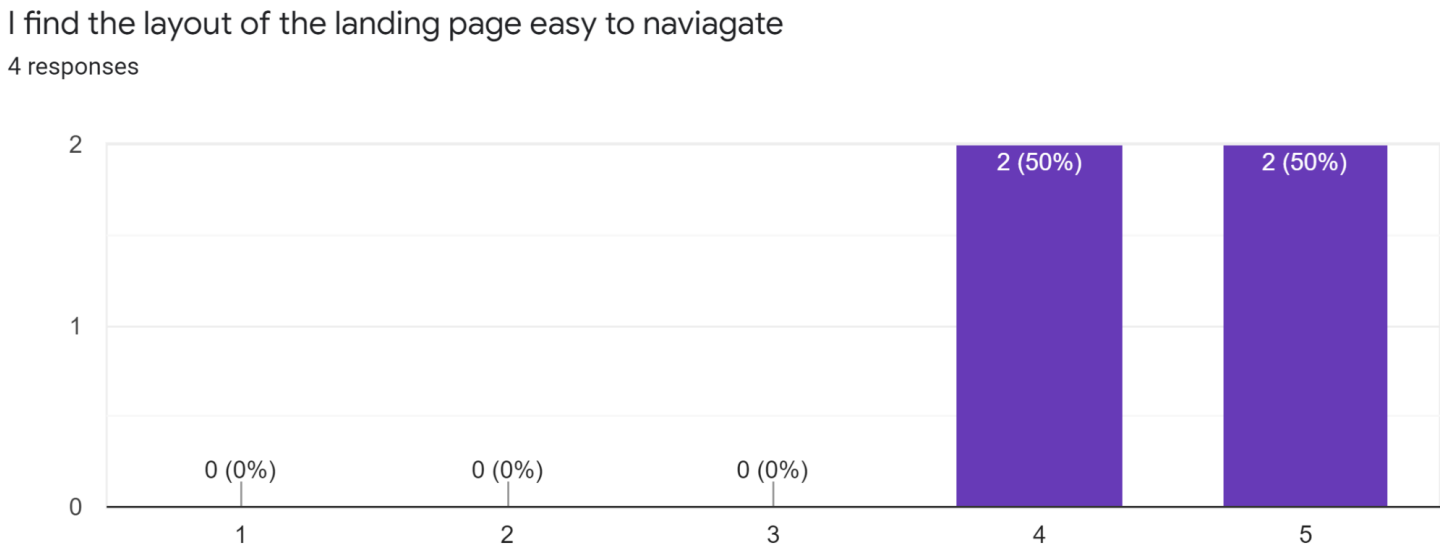


Figure 10: *User Testing Iteration 1 = "I find the layout of the landing page easy to navigate"*

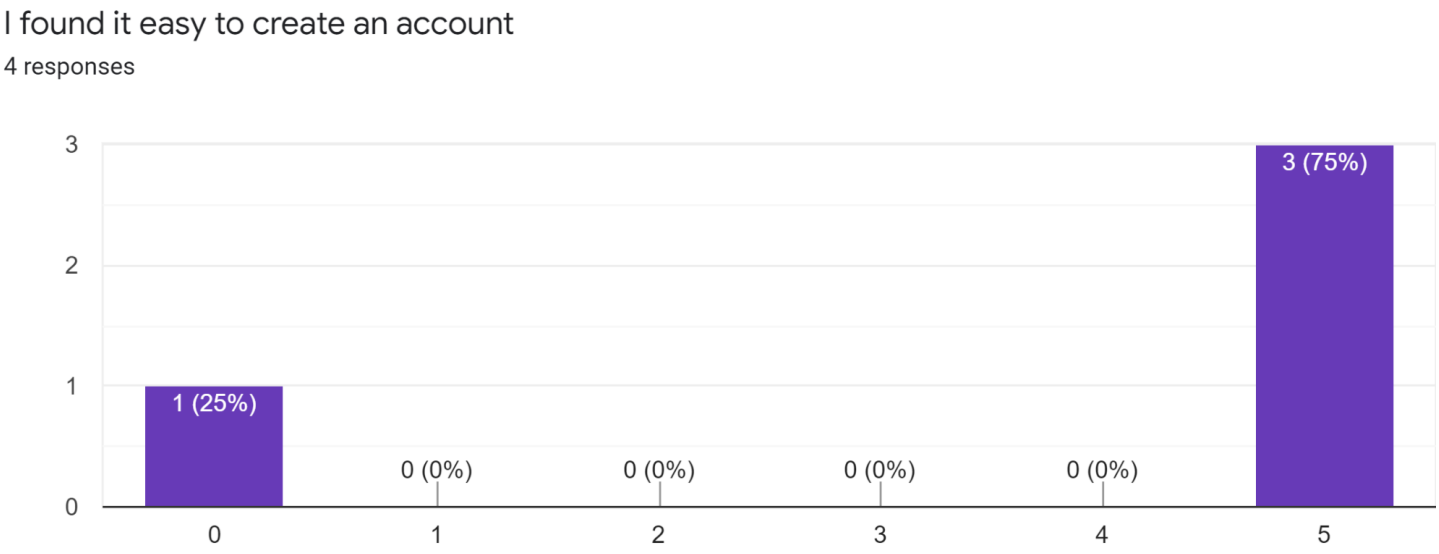


Figure: 11: *User Testing Iteration 1 = "I found it easy to create an account"*

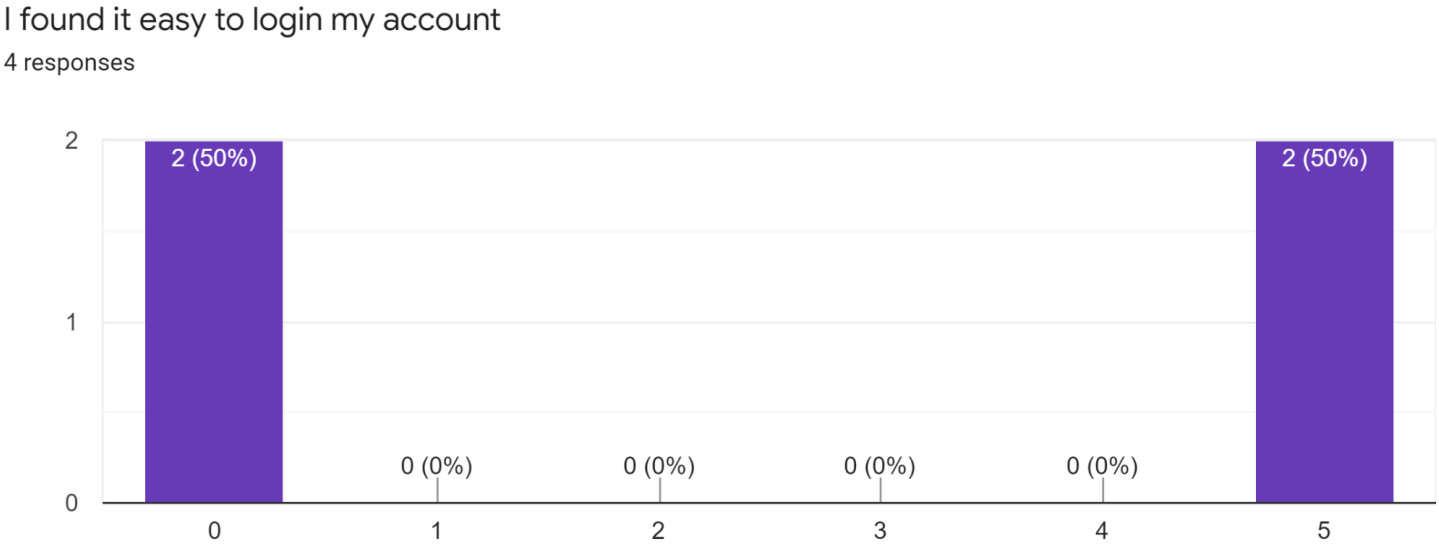


Figure 12: *User Testing Iteration 1 = "I found it easy to login to my account"*

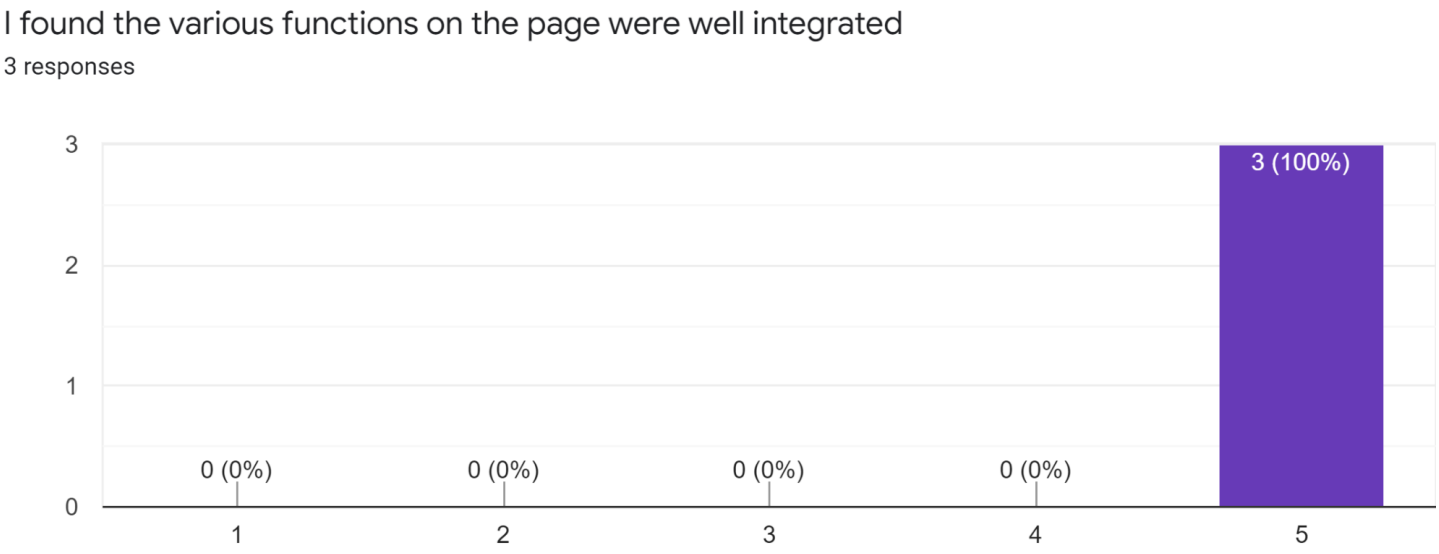


Figure 13: *User Testing Iteration 1 = "I found the various functions on the page were well-integrated"*

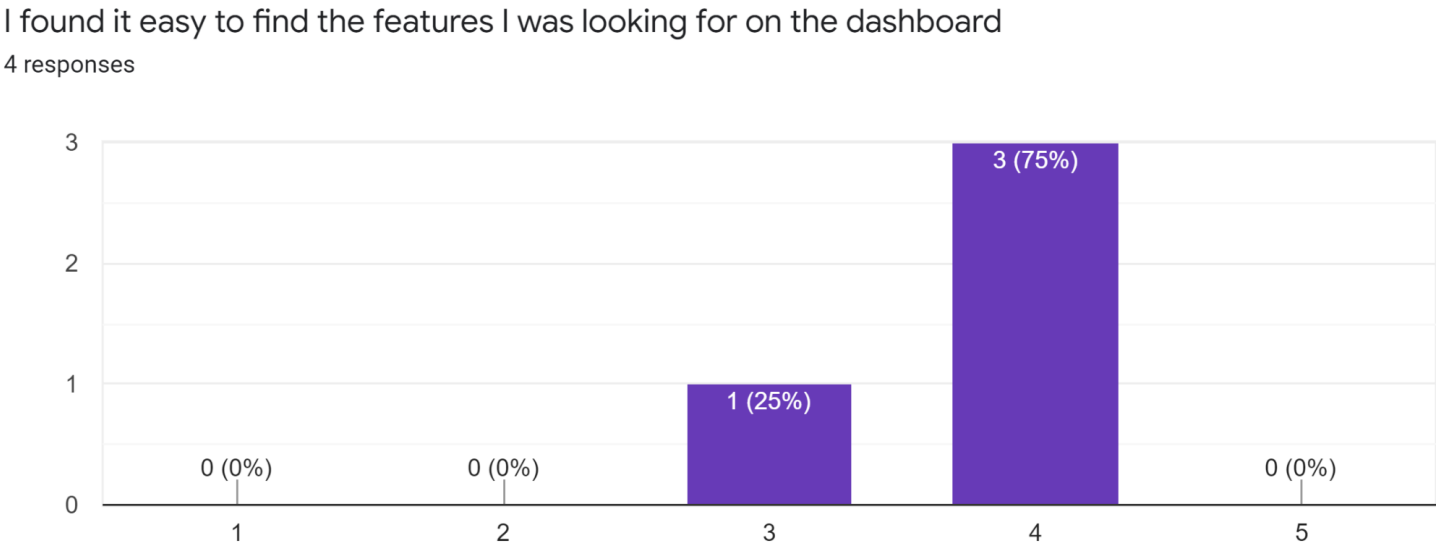


Figure 14: *User Testing Iteration 1 = "I found the features I was looking for on the dashboard"*

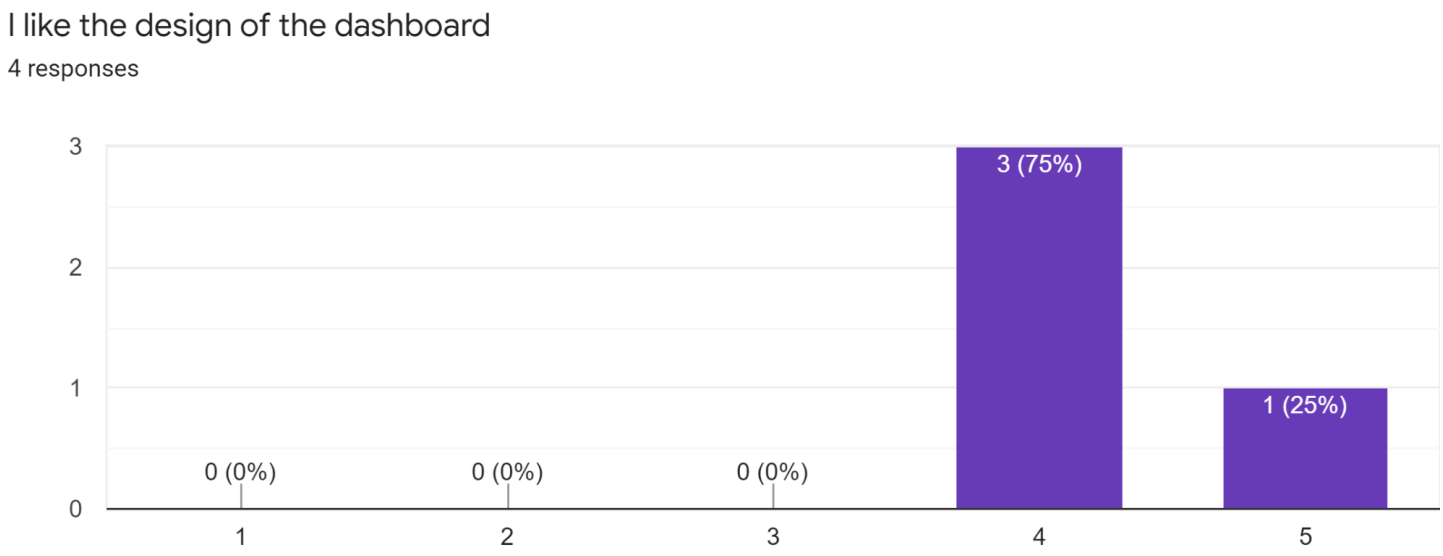


Figure 15: *User Testing Iteration 1 = "I like the design of the dashboard"*

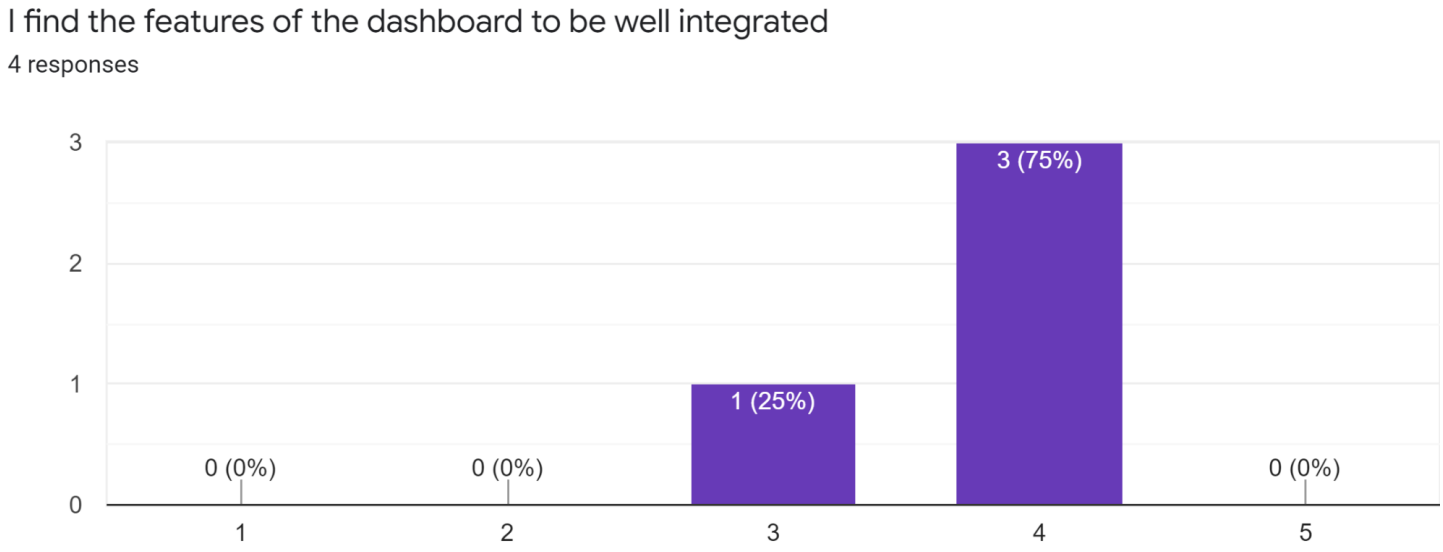


Figure 16: *User Testing Iteration 1 = "I find the features of the dashboard to be well integrated"*

Across the board testers thought that the design and layout of the website's landing page were "enjoyable" with the average score being 4.5 along with the landing page being easy to navigate and also carrying the same average score of 4.5. This data indicates that the changes made from the high fidelity toward the new production prototypes were successful in terms of design. In all cases, testers rated 5 for ease of logging into / creating an account (accounting for a user tester error by filling in the question). This shows us that users are able to navigate the site smoothly as they could find the correct buttons and paths of action to take easily which is also backed by the average score of 5 for integration. Users were asked about any changes they would make to the landing, and

overall people stated that they would like to see a change in the colour by adding a different colour on top of the white or black depending on light mode, along with adding some other elements to fill space.

Regarding the dashboard, users gave an average score of 3.75 about being able to find different elements on the page, with the same score being given about page integration. This indicates to us that some reorganising needs to take place on the dashboard to allow users to be less “cluttered”. The design got an average score of 4.25, similar scores to the design of the landing page. From this, we can make the assumption that the slight decrease was due to the issue of “clutter”. Comments from the testers touched on changing the spacing of the page with two testers giving specific movements of elements in order to create more space. These changes included moving the location of the notification box along with increasing the size of the trading box allowing for more assets to be seen.

Some points of criticism to be made regarding testing is that there were not as many testers as was desired. By having more testers it would be possible to have a larger range of data, along with the possibility for more aspects of design changes or functionality ideas for future iterations. Another issue regarding the first iteration is the errors of numbers not matching up on the form, this could be from a form error or people missing the questions. For the next iteration of testing these issues would be approached by potentially increasing the sample size along with improving the form method to get feedback.

Functionality Testing Data

Going into the later stages of testing, the objective was to use the information gained from the previous testing and to implement aspects that we found to be constructive in order to better the web application on their own recommendations. In this run, this team took into account any errors that were made before, such as any miscommunication on the feedback form, along with increasing the number of testers from 4 to 7. To make sure the data is not too far skewed, the original 4 testers were asked to give feedback along with the new first time testers.

Overall - second iteration

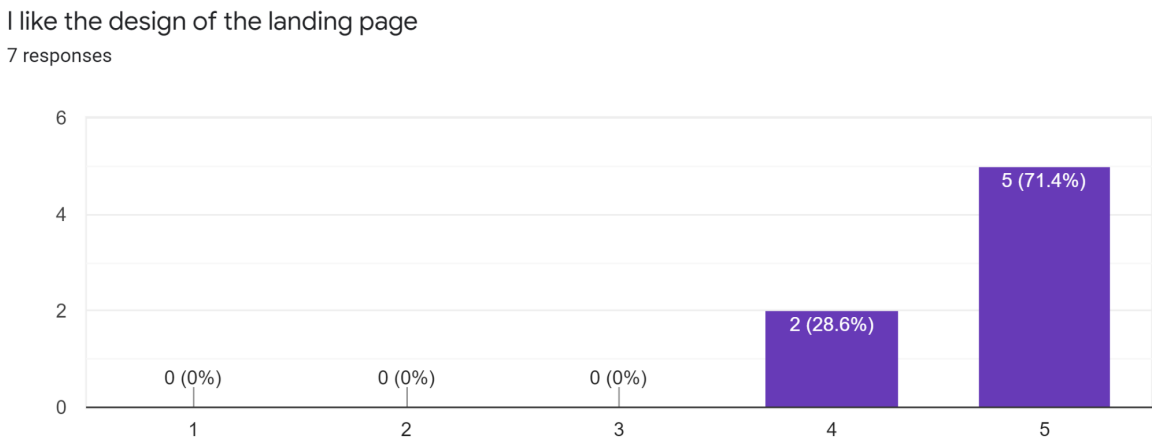


Figure: 17: *User Testing Iteration 2 = "I like the design of the landing page"*

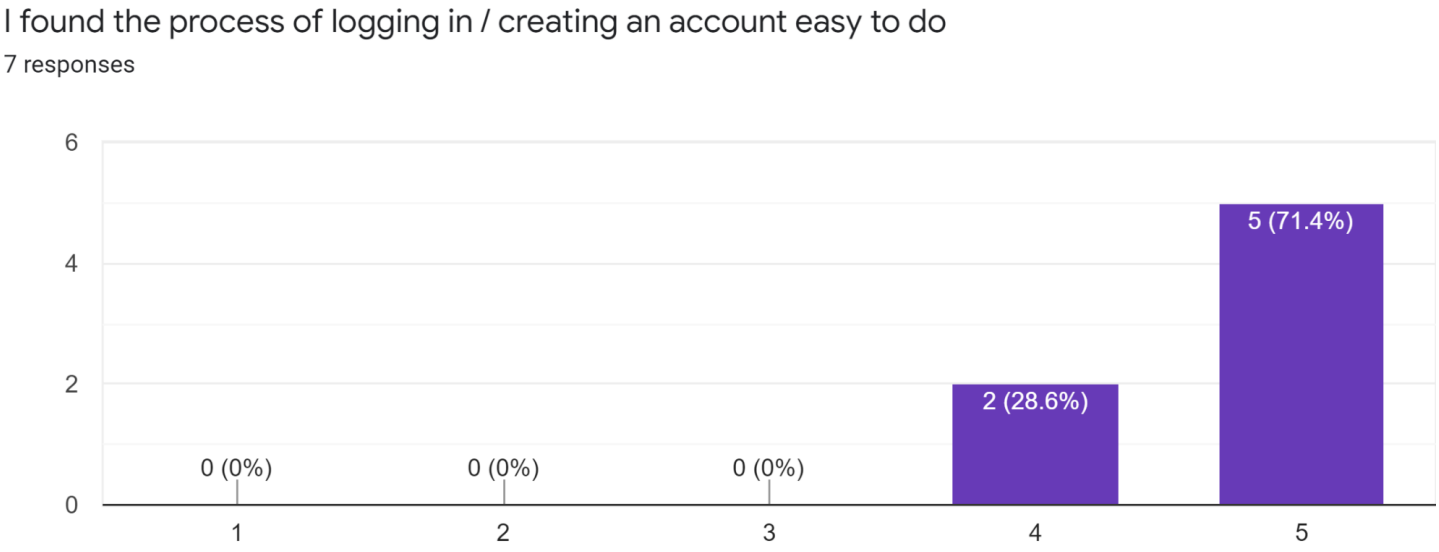


Figure 18: *User Testing Iteration 2 = "I found the process of logging in / creating an account easy to do"*

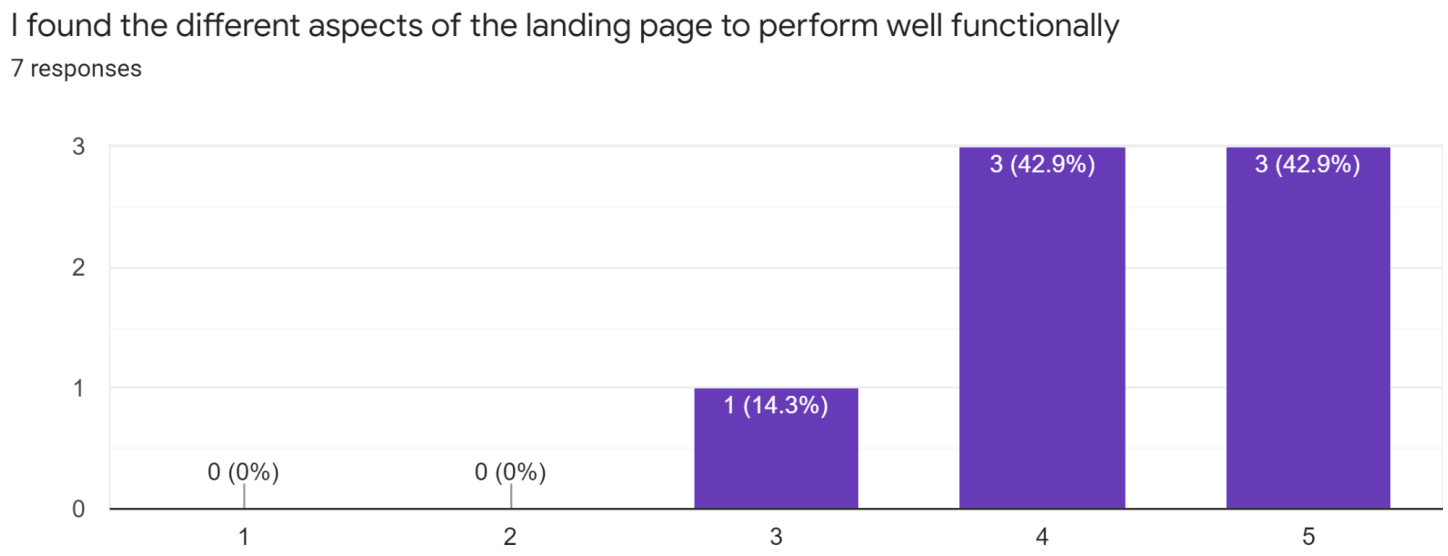


Figure 19: *User Testing Iteration 2 = "I found the different aspects of the landing page to perform well functionally"*

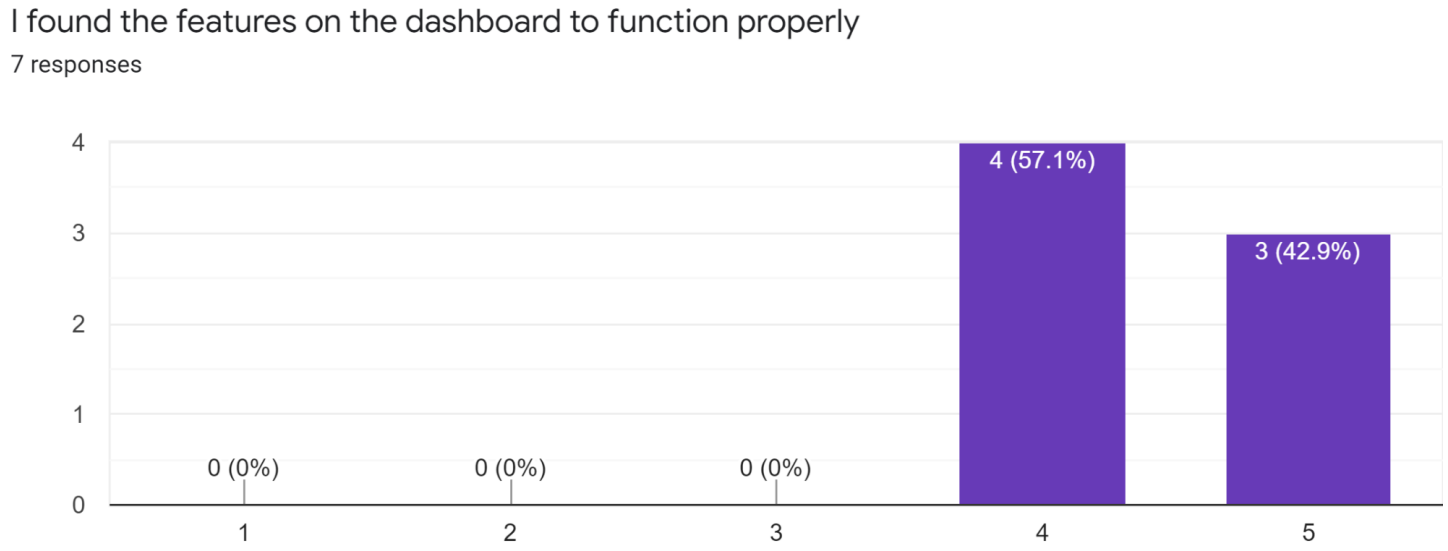


Figure 20: *User Testing Iteration 2 = "I found the features on the dashboard to function properly"*

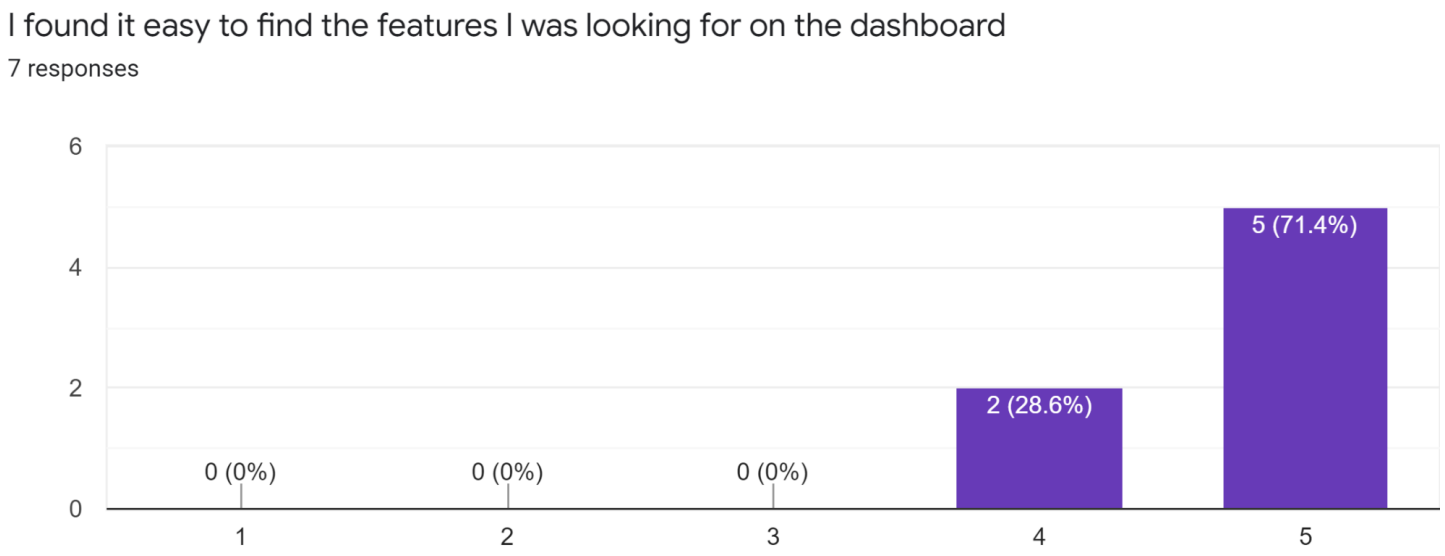


Figure 21: *User Testing Iteration 2 = "I found it easy to find the features i was looking for on the dashboard"*

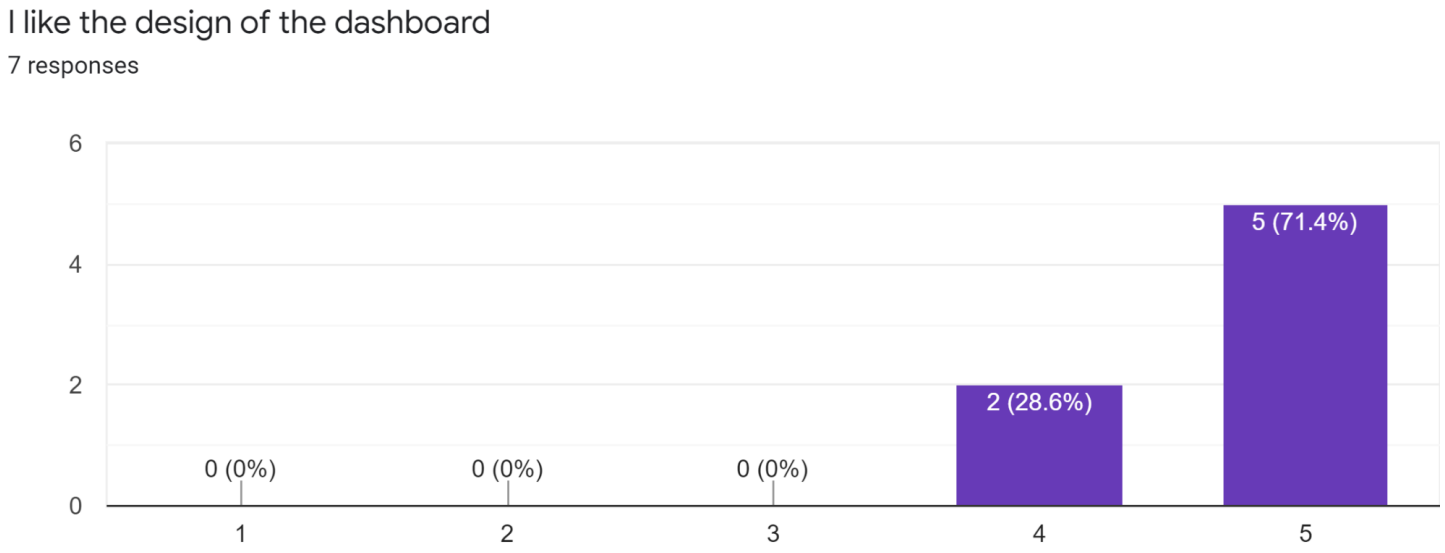


Figure 22: *User Testing Iteration 2 = "I like the design of the dashboard"*

The main changes that we made this time for the dashboard was to include more colour which was highly suggested. Upon the addition of a large circle of colour it can be seen that testers were more satisfied by the changes with the design scoring a 4.71 compared to the previous 4.5, along with this there were no comments about any design changes from testers. For users trying to login / create an account, this team found that the majority of users (5) were able to go about the process without any issues at all, the other 2 had issues with logging into the system after registering, requiring a refresh or to create another new account and try again. The

scores based on functionality decreased because of this, with an average score of 4.2, with the lowest score being a 3 from one user who had multiple issues. For future iterations of testing more focus regarding aspects of the dashboard will be focused on user functionality and streamlining gaining access to the dashboard.

On this iteration of testing, we can see a clear increase in scores related to the dashboard. This can be seen from the overall increase in scores regarding users' enjoyment of the dashboard design, with the score going from an average of 4.25 to 4.7. Alongside this, users had a greater response to being able to find different features within the dashboard, this is seen in the averages with values going from 3.75 to the much higher 4.71 showing a clear favour in the new layout of the improved dashboard. For functionality, it can be seen there is a larger proportion of values towards the 4 regions with an average of 4.42. Users commented that they had issues with the different features not working properly on the page, however this can be explained through the issues with development. Due to delays in development we wanted to have elements for the users to interact with whilst development still takes place. For future testing iterations we wish to have full functionality for elements.

Heuristic evaluation

It was decided to use heuristic evaluation because of the lower number of testers that were present in the initial design feedback. By conducting heuristic evaluation it was straightforward to get immediate feedback from the in-house team, however, this also opens the door for any biases that this team might have towards our own product. For the testing, Jakob Nielsen's evaluation guidelines were used (Nielsen).

1. Visibility of system status

The system should always keep users informed about what is going on, through appropriate feedback within a reasonable time.

2. Match between system and the real world

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

3. User control and freedom

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

4. Consistency and standards

Users should not have to wonder whether different words, situations, or actions mean the same thing.

5. Error prevention

Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

6. Recognition rather than recall

Minimise the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

7. Flexibility and efficiency of use

Accelerators — unseen by the novice user — may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

8. Aesthetic and minimalist design

Dialogues should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

9. Help users recognize, diagnose, and recover from errors

Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

10. Help and documentation

Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

	1-2:
	3-4:
	5:

Heuristic -->	Visibility of system status	Match between system and the real world	User control and freedom	Consistency and standards	Error prevention	Recognition rather than recall	Flexibility and efficiency use	Aesthetic and minimalist	Help users recognize, diagnose, and recover from errors	Help and documentation
Scale for evaluation	3	5	3	3	5	4	4	5	5	2

Table 3: *Colour-coded scheme for heuristic evaluation*

Initial Design

For our initial design, we decided to use the previous prototypes that we had made in the concept report and go about bringing the ideas to life, with a few conceptual changes at first as to create a better framework for the application to be built upon. The team decided to stick to these initial prototype designs as we had good feedback from initial testing and felt that no further iterations would have to be made, instead, any later changes to be made depending on later testing and feedback.

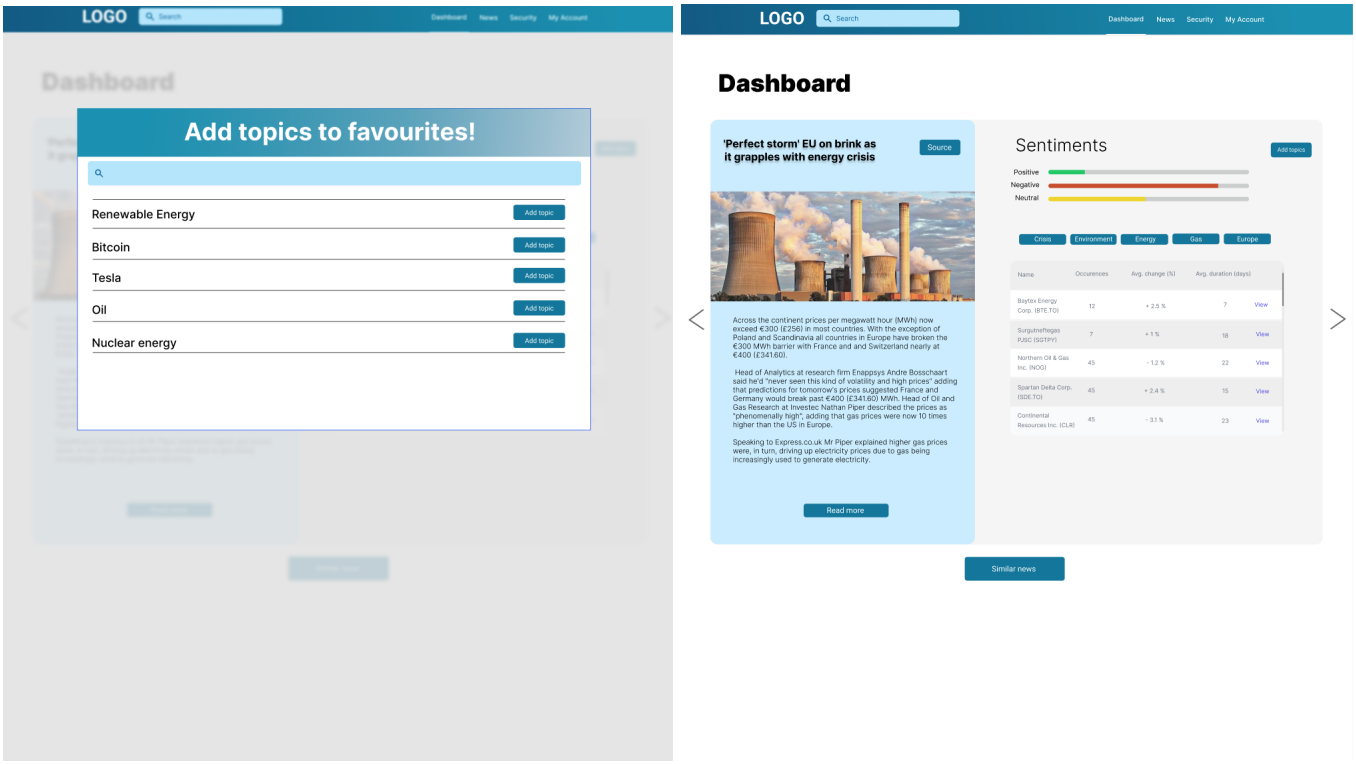


Figure 23: *High-fidelity prototypes*

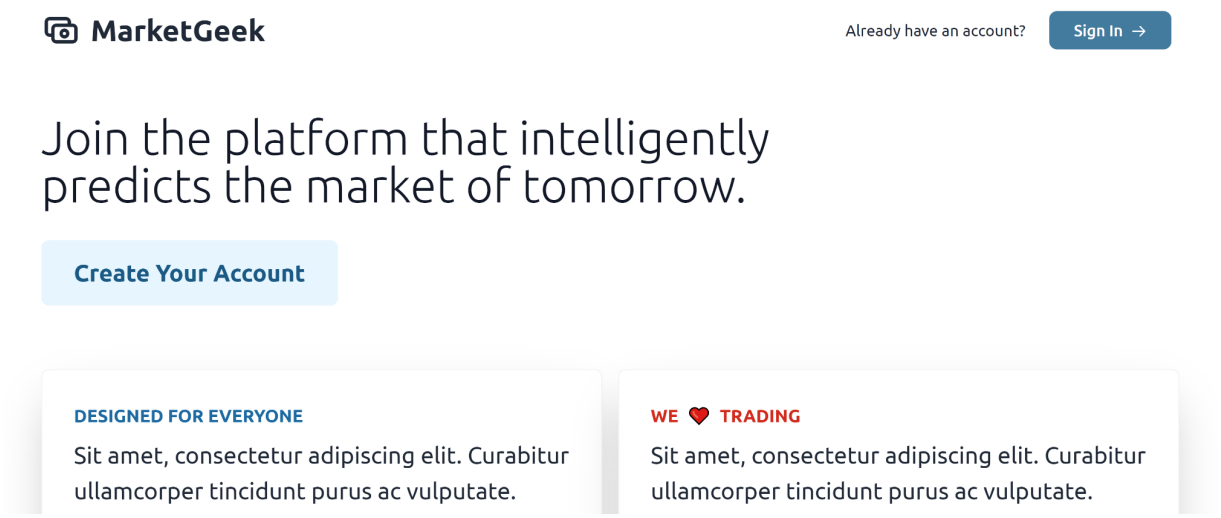
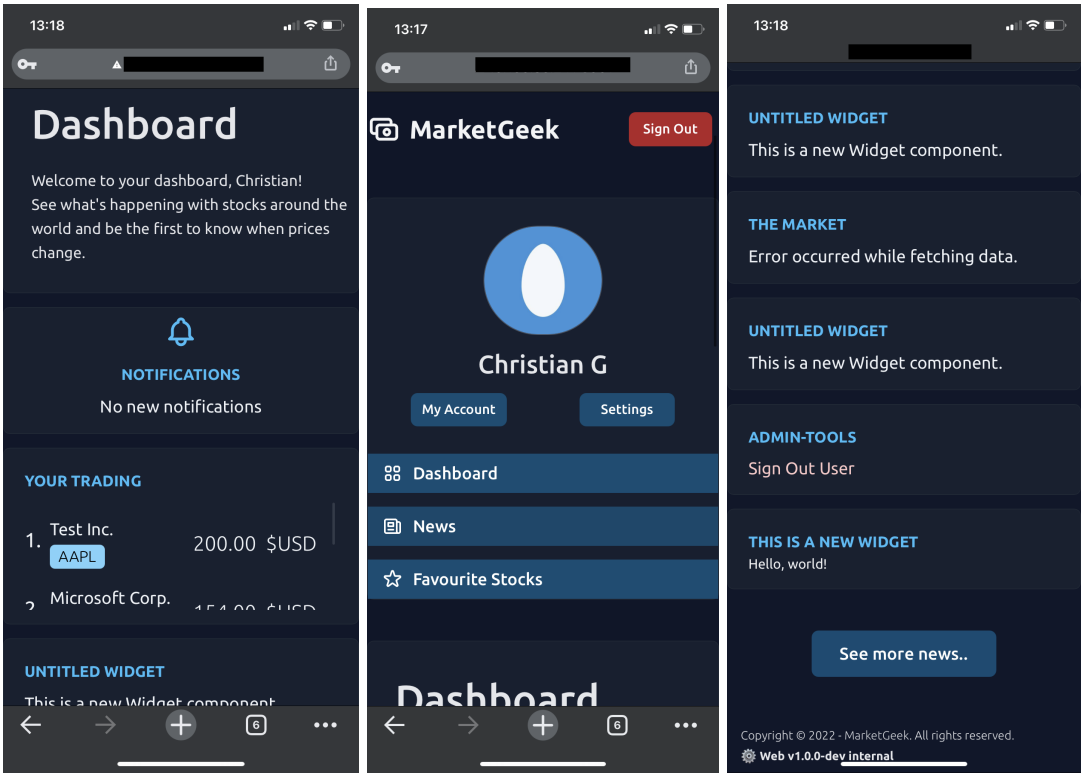


Figure 24: *First iteration of production prototype (Desktop) (Light Mode)*



Production prototype, first iteration

From the first production prototype we decided to stick closely to the high-fidelity prototypes by having the same blocking structure as seen in the mockups, for example, the design of the navigation bar has been refined to look sleeker with a better colour gradient than what was shown in the prototypes. The overall colour scheme has been adapted from the high-fidelity based on user feedback from early testing to provide for a more “sleek” colour scheme based on familiar designs such as the design of apple’s user interfaces as this makes people feel more familiar with the product (Zuma). The new colour scheme still sticks to our original choices in which we found the primary design around a blue colour gives more confidence in technical applications from initial research (Mone).The dark mode which can be seen in figure 25 allows for a nice contrast to the light mode,creating an effective polarisation of the different elements on the web application, which we found from testing (Biriukov et al.) allows users to find the different features implemented, especially when considering the mode to be used more in harder lighting conditions, such as night time or in the sun, allowing for easier navigation.

Secondary Design

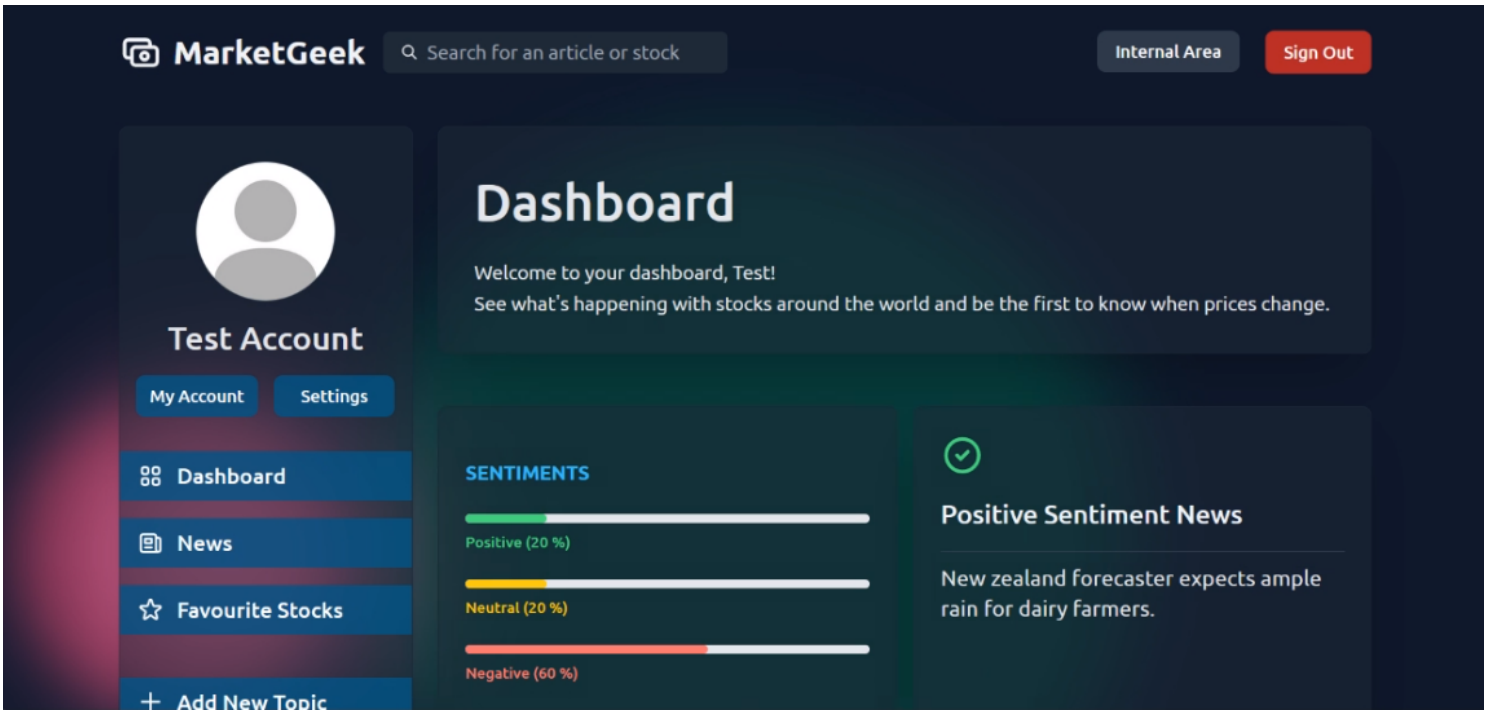


Figure 26: Second iteration of dashboard (Desktop) (Dark Mode)

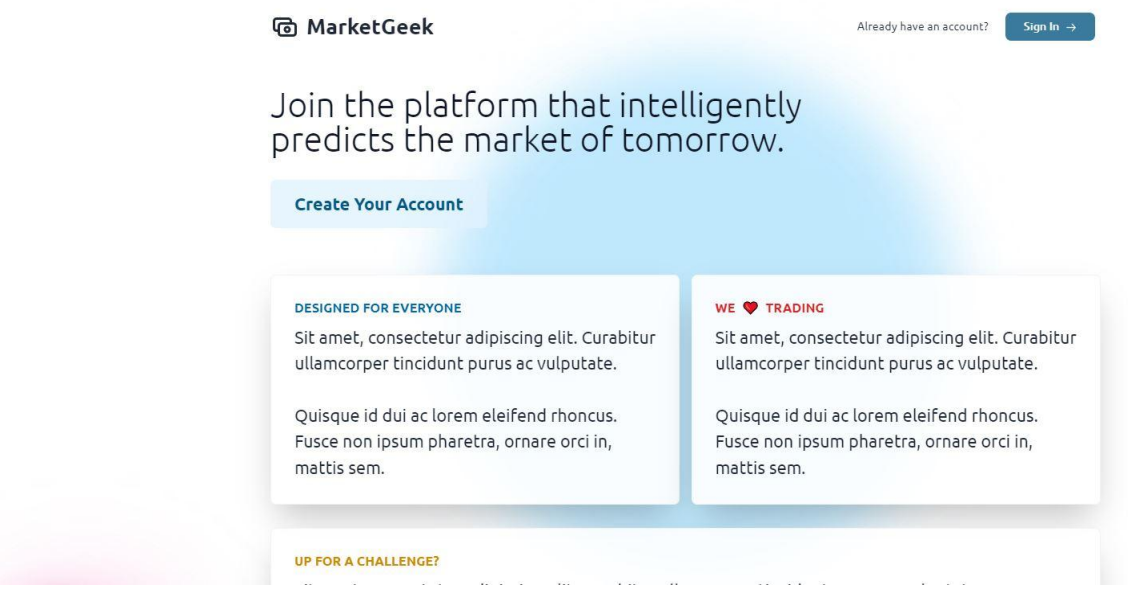


Figure 27: *Second iteration of landing (Desktop) (Light Mode)*

For the second round of production design, we took the constructive criticism from the first round of testing. Regarding the design, we found that most of the feedback for the landing page was centred around not having enough variation of colour, and users said they found it “plain” or “boring”. The team decided to add something simple that is also able to draw user attention away from the empty spaces, in this case a pulsating circle. The team stuck to the colour blue as of the aforementioned favouritism towards the blue (Mone 2002) for the light mode, and for dark mode took the approach of moving towards a heavy turquoise colour with hex values between blue and green as to keep to the styling whilst also not blinding users.

Feedback for the designs of the dashboard had similar criticism about the colours so we followed the same design changes by adding the circles of colour to improve on the decal. Another major criticism that we got from testers was that the dashboard was too cluttered, with users being unable to find different components on the page (as can be seen in Figure 27). To fix this issue we decided to move the notification box and user trading box out of the way and instead replace them with the sentiment table and a news box directly underneath.

From the second round of testing, we had a positive level of response to these changes. This can be seen in Figures **26** and **27** with the average scores of users' enjoyment of design increasing after each of the separate changes, along with a decrease in users being unable to find different elements.

Components of the software

This software consists of a set of three main components: the web app, an application that includes an API for handling the user authentication and the back-end that is analysing stock data using ML (Machine Learning).

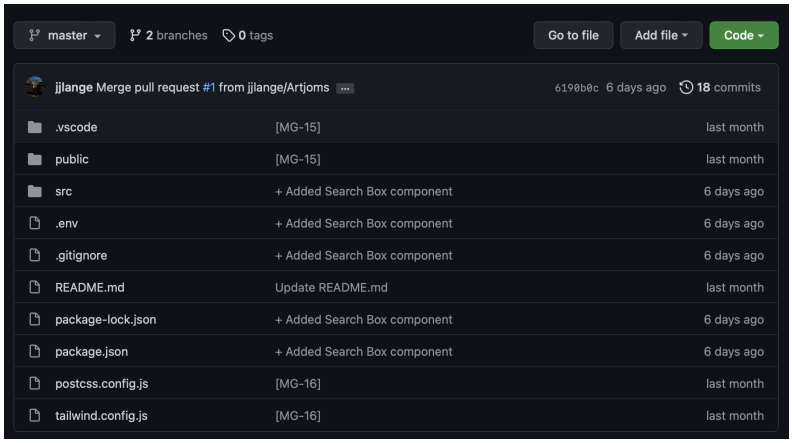


***Figure 28**: GitHub repository for front-end*

Presentation layer

The web application is the main presentation layer of this project and runs on devices that support browsers, such as Google Chrome, Microsoft Edge or Safari on their respective latest versions. They all need to have JavaScript enabled. Clients can use the application by opening the web address of MarketGeek in the URL field of their web browser.

Web Applications are often broken down into several HTML, JavaScript and CSS files where clients need to send additional HTTP requests for each file. This might result in slower site loading times, a less structured code-base or code that's not in use anymore but still being loaded by the client.

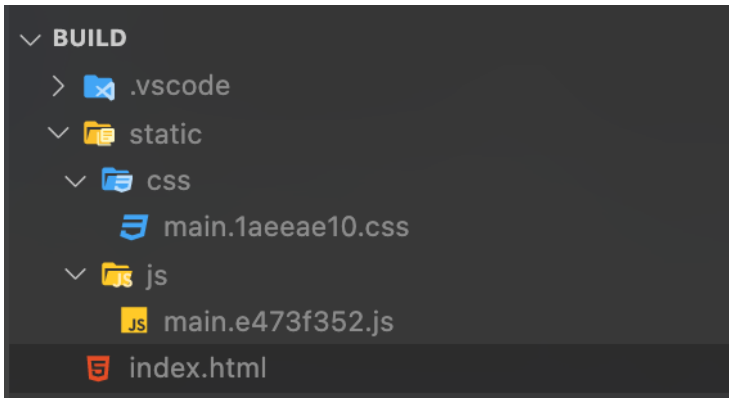


***Figure 29:** File/folder structure of the build*

React takes a different approach where compiling the whole web application ends up in a single HTML file that includes one bundled JavaScript file. This file includes all parts of this User Interface.

React introduced a new way to define elements in views and allows using logic inherently coupled with this User Interface. HTML elements can be displayed using their web browser, which were rendered by React.
The bundled JavaScript file exports all required HTML elements of a view into the HTML file.

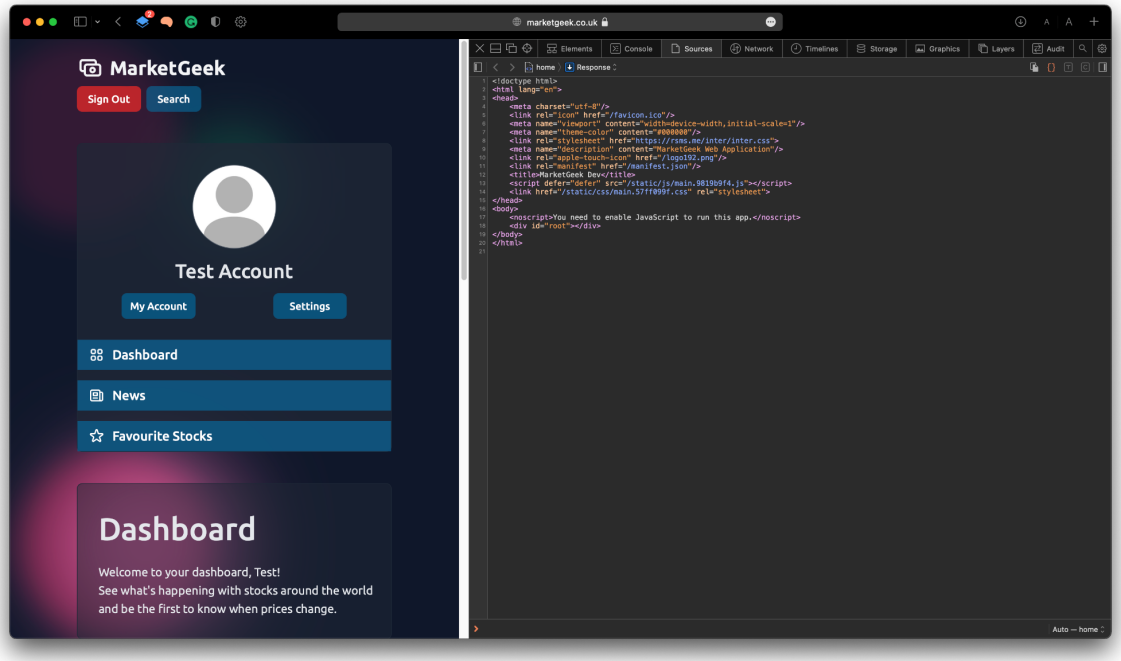


Figure 30: *Web application and its HTML code*

The web browser of a client downloads all required files (**index.html**, the **bundled JavaScript file** and **CSS files**) from the server. The bundled JavaScript file contains the React application in a compiled format that uses short function and class names, variables, etc.

The application can be compiled using the **npm run build** command in the root folder of the project. This command is built into the React npm framework.

Structure and Implementation

The React application takes a simple View-Controller approach as seen in **Figure 31**. The main entry point of the application is in the index.js that creates the ReactDOM which allows us to render JSX template files and connect those with the according root div in the index.html file.

The pages that are ending with the file extension **.jsx** are located in the **Views** directory, so the team decided to put all individual large components in the subfolder "**Components**". This gives us the freedom to modify existing components throughout many pages at once, manage our code in more files and therefore avoid having to work with large single file codebases.

The **Utils** folder contains utility functions for the user management system. This system allows us to retrieve session information of the current client and authenticate / check tokens by sending requests to our custom API Server. JSON Web Tokens are part of the security mechanism behind our software and respective functions are to be found in the **Utils**.

In order to preview information about stocks, sentiments or protecting pages from users that are not signed in, we decided to put the following methods in a new folder called "**Hooks**". The team initially had methods for finding and retrieving user information in a **findUser.js** class (as seen in the right **Figure 31**) but then decided to include the method in our User.js file to keep our structure and code more organised.

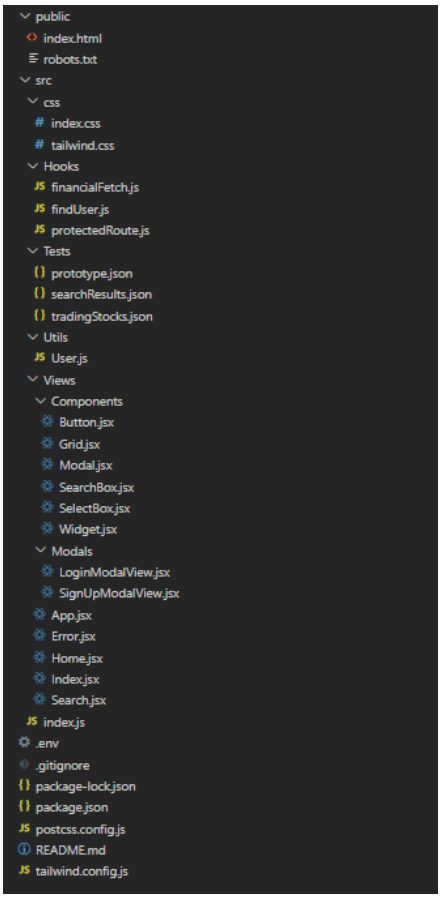


Figure 31: *File structure*

Some content in our web application is relying on JSON data from the back-end team, our custom API Server and the Polgyon.io API. The team created multiple JSON test files to prototype the functionality behind our system in a test environment that would allow us to add new data and modify existing data quickly. The test files were initially JSON files for testing Accounts, results on the search page (**still being used**) and data to show popular stocks on the Dashboard. After implementing different API requests in our application, we slowly removed the unused JSON files from our project and only decided to keep the search results JSON since no requests are being made to the back-end server at this point. All test files are accessible using a GET request to the Test folder which means that the folder is available in public build versions of the project.

The **tailwind.config.js** and **postcss.config.js** contain all the configuration options for their respective libraries throughout the project. Other configurations, such as the A**PI Server IP** address can be changed in the .env file in the root directory of the project. The **package.json** contains all necessary information (such as the name, licence or author) as well as the installed packages.

Browser Support

Late March 2022

This is a list of the currently supported browsers on different devices, any browser that is lower than the given version does not have JavaScript support and will not be able to load the web site. This team does not guarantee any official support for browsers in this list, instead, a separate list with browsers that have actively been tested was created.

Supported

Browser	Version
IE (Internet Explorer)	**Not supported**
Edge (Microsoft Edge)	79 - 99
Firefox	67 - 100
Chrome	67 - 102
Safari / Safari on iOS & iPadOS	11.1 - 15.4
Opera	50 - 83
Opera Mini	**Not supported**
Android Browser	99
Opera Mobile	64
Chrome for Android	99
Firefox for Android	96
Samsung Internet	8.2 - 16.0

Table 4: *supported browsers for this application*

Recommended

Late March 2022

The web application has been tested in different browsers using popular Operating Systems. The team used real devices (Android, iPhone, iPad), simulators (Xcode Simulator) and online-tools such as BrowserStack (reference!).

Browser	Operating System	Version
IE (Internet Explorer)	Windows 10/11	**Not supported**
Edge (Microsoft Edge)	macOS, Windows 10/11	79 - 99
Firefox	macOS, Windows 10/11	67 - 100
Chrome	macOS, iOS & iPadOS, Windows 10/11	90 - 102
Safari	macOS, iOS & iPadOS	13 - 15.4

Table 5: *Recommended browsers for this application.*

The API server

This layer of the application is responsible for handling requests between the web application and the MongoDB database.

The API Server is a Node.js application that requires multiple packages to work properly (such as Express, MongoDB, bcrypt and others). These packages can be installed using nodes pre-installed package manager “npm”. A list of the required Node.js modules
are listed below in (**Figure 32**).

Express is used to handle GET & POST requests and to run a HTTP server on a specific port. MongoDB is responsible for the communication between the API Server and MongoDB collections
to retrieve, validate and modify user information securely.

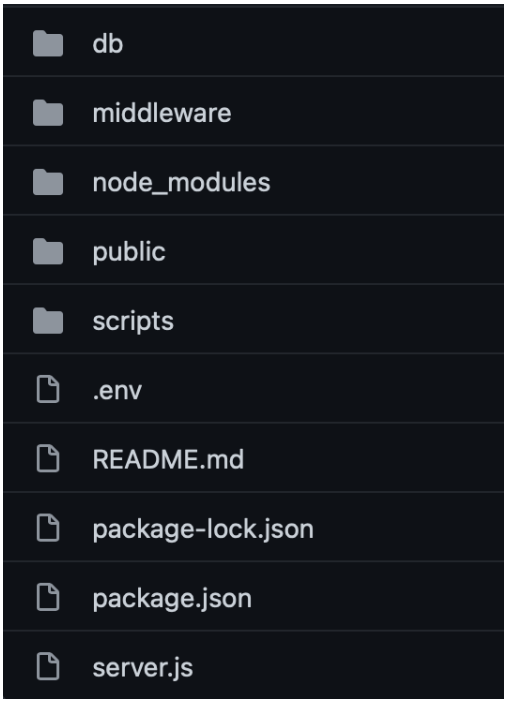


Figure 32: *API Server folder structure*

The main entry point of the application is in the server.js file that handles the specific requests to the server as seen in {figure/table} and connects to the database using credentials set in the .env file in the root folder of the project. It was decided to separate the API calls to the MongoDB server and specific collections in a database driver file and export methods that can then be used in the main server.js with incoming requests.
Figure {figure_num}: server.js file and database driver connection

```
/* ... */

const db = require('./db/dbCon');
db.connectDB()

/* ... */
```

```
/**
 * Method to connect to the database and set the database variable
 **/
connectDB: function() {
    client.connect(function(err) {
        if (err) {
            console.log('SERVER: Error connecting to database!')
            return callback(err)
        }

        db = client.db('marketgeek')
        console.log('SERVER: Database connected!')
    })
},
```

Figure 33 (left): *server.js - connecting to database,* ***Figure 34 (right):*** *connectDB method in dbCon.js*

The database file exports methods such as createUser, deleteUser, authUser, or checkUserExists. Most of the methods share the same layout and consist of a MongoDB collection call that returns a result and error, these are both handled within the function and return a callback to the caller.

The parameters (aside from the callback argument) are mostly identical to the arguments that are required to make the API call.

JSON Web Token

Authentication using JSON Web Tokens is handled on the server-side API server when requests are made, this is done using several middleware methods.

JSON Web Token is an open standard to securely transmit information (as JSON objects) between clients and servers online. (“JSON Web Token Introduction - jwt.io”, n.d.)

```
/**
 * Method to verify an existing JWT token
 * @returns HTTP Status
 */
authToken: function(req, res, next) {
  const authHeader = req.headers['authorization']
  const token = authHeader && authHeader.split(' ')[1]

  if (token == null) return res.sendStatus(401)

  jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (err, decoded) => {
       if (err) return res.sendStatus(403)
       req.user = decoded
      next()
  })
}
```

Figure 35: *authorisation token using JSON*

The authorization token is created when a client successfully authenticates on the web application.
Before rendering a page, the web app sends a request to the API Server to request import data from the user.

It is stored in the client's browser / transmitted within the HTTP request allows us to verify it on the server-side using the checkToken route (see list of endpoints under “API Server”). The verification is done using an Access Token Secret that is defined in the .env- configuration file within the API server root directory.

We used the information behind the Access Token to retrieve the UUID of the user (using **returnUserId**), this allows us to find a result with the same ID within the users collection and returns all necessary users’ information (such as their **name, email address or gender**).

```
app.get('/api/v1/user/checkToken', authToken, (req, res) => {
    let user = {
        _id: returnUserId(req),
    }

    db.getDB().collection('users').findOne({"_id": new mongo.ObjectId(user._id)}, function(err, result){
        if (err) {
            console.log('SERVER: Error getting user')
            console.log(err)
        }

        if(result) {
            let response = {
                user: result,
                message: "token_valid"
            }
            res.status(200).send(response)
        } else {
            console.log("SERVER: Error getting user")
            console.log(err)
            res.status(500).send(err)
        }
    })
})
```

Figure 36: *checkToken endpoint*

Sign up and sign in

The Sign In and Sign Up pages on our project have been implemented with React components and using Tailwind CSS to style those. In the figures below you can see the final view of the Sign Up form that allows the end-user to create a personal account for secure and reliable access on our web app.

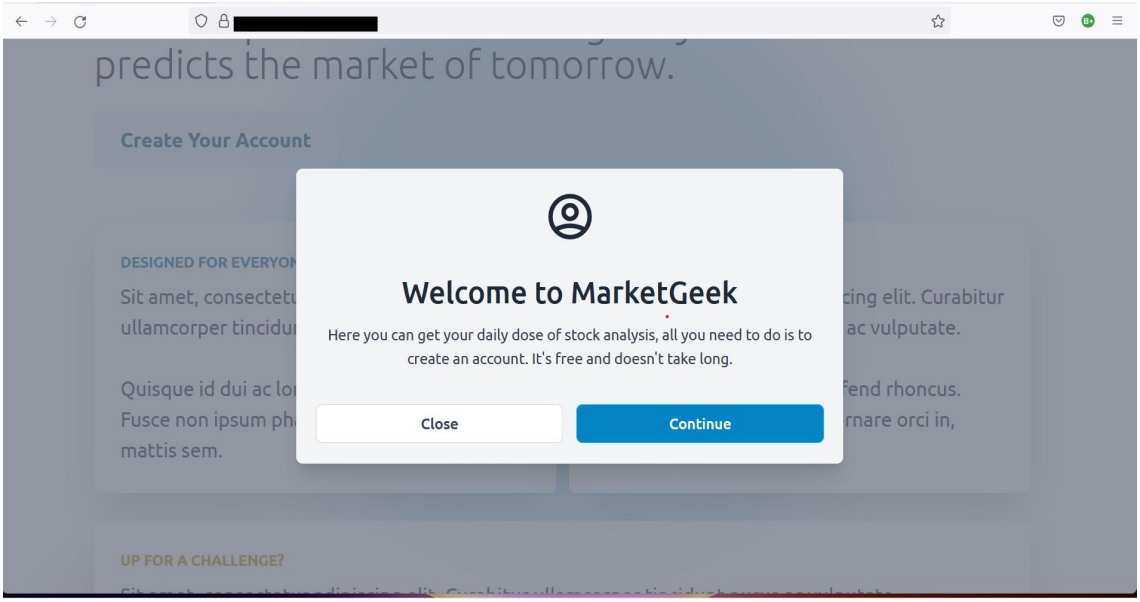


Figure 37: *Signup disclosure*

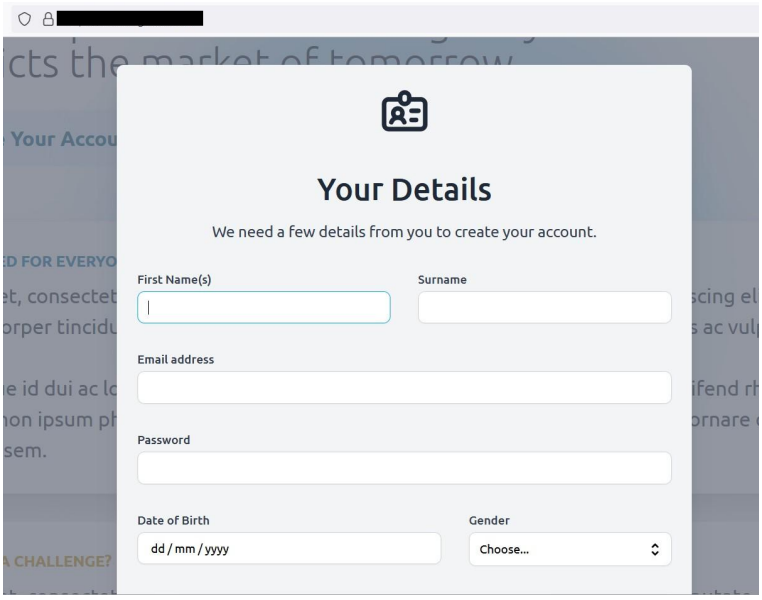


Figure 38: *Registration disclosure*

This page has been implemented using separate React components, which has then been imported and used in the **Index.jsx** file.

```
import LoginModalView from "./Modals/LoginModalView";
import SignUpModalView from "./Modals/SignUpModalView";
```

Figure 39: *login and sign up components*

Both components are saved and can be modified in the project's “Modals” folder. The view LoginModalView is responsible for retrieving information that the user can enter into input fields. The function takes on arguments (props) that are being used in the event handler of the text fields as seen in **Figure 40**.

```
                    <input
                    id="email"
                    name="email"
                    type="email"
                    autoComplete="email"
                    className="block w-full px-3 py-2 border border-gray-300 dark:border-gray-600
rounded-lg shadow-sm placeholder-gray-400 focus:outline-none focus:ring-sky-500 focus:border-sky-500
dark:bg-gray-700 sm:text-sm"
                    value={props.email}
                    onChange={props.onChangeEmail}
                />
```

Figure 40: *LoginModalView parameters*

The view LoginModalView is responsible for retrieving information during the sign up process that the user can enter into input fields. Similar to the **LoginModalView**, it also accepts the “props” - properties as seen in **Figure 40.** The Sign Up form implements an additional component called SelectBoxthat handles the Gender Selection in the Sign Up Modal. An id represents the selection for each option/gender.

Both views make use of the **onChange** event handler, which detects a change in the input field by the user and updates a specific variable to store the input.

For endpoints related to the sign up and sign in process, see more information about them in the **endpoints table** under “**The API Server**”.

```
// Imports
import SelectBox from '../Components/SelectBox'

// Content view for the Sign Up modal
const SignUpModalView = (props) => {
    const genderChoices = [
        { id: 1, name: 'Male' },
        { id: 2, name: 'Female' },
        { id: 3, name: 'Other...' },
    ]

    return (
        <div className="mb-12">
            <div className="flex space-x-8">
                <div className="w-full">
                    <label htmlFor="firstName" className="block text-sm font-medium text-gray-700
dark:text-white  pt-8 text-left">
                        First Name(s)
                    </label>
                    <div className="mt-1">
                        <input
                            id="firstName"
                            name="firstName"
                            type="name"
                            autoComplete="name"
                            className="appearance-none w-full px-3 py-2 border border-gray-300
dark:text-white dark:border-gray-600 rounded-lg shadow-sm placeholder-gray-400 focus:outline-none
focus:ring-sky-500 focus:border-sky-500 dark:bg-gray-700 sm:text-sm"
                            value={props.firstName}
                            onChange={props.onChangeFirstName}
                        />
                    </div>
                </div>
```

***Figure 41**: SignUpModalView component*

Our project configuration file (or a .env file) is a simple text configuration file for controlling web-application environment constants. In the majority of applications, the environment between Local, Staging and Production will not change. However, in our project we had instances in which configurations were altered. For instance, API keys are changed many times, same as the local host access port.

An environment variable is made up of a name/value pair, and any string, integer or boolean may be created and available for reference at a point in time.

Figure 42: *.env file (web application)*

For security reasons, the protected information within the .env file shall not be disclosed in any Git repository. This is why we avoided uploading a public version with sensible information to the repository.

Unfortunately the .env configuration brings another problem: Any information in the .env file is readable through modern browsers by searching through the build JavaScript file. This problem is described in the technical issues section on the front-end side.

Modules

This list shows required modules to build and run the API Server. The installation has been tested under macOS Monterey 12.1, Windows 10 and 11 (most recent builds) and Ubuntu Server 18.04 LTS.

Package Name	**Description / Use-case**
console	Build-in module to clear the console before logging info messages and requests.
express	Minimal web framework that provides APIs for HTTP utility methods
http	Low-level module that allows to serve websites and transfer data over the HTTP protocol
body-parser	Middleware that parses incoming requests and information available under the req.body property
mongodb	Official MongoDB module to communicate with the database server using methods
jsonwebtoken	Implementation of JSON Web Token in JavaScript for signing and verifying tokens.
dotenv	Module to read a .env configuration file
cors	Middleware to configure web server cors such as allowed methods, headers and origin
bcrypt-nodejs	Implementation of bcrypt hashing in JavaScript to securely store passwords in the database

Table 6: *modules required for API server*

Endpoints

API Endpoint	Method	Description	Arguments	API Key
/api/v1/users	GET	Returns a list of users from the collection “users”.	-	True
/api/v1/user/create	POST	Inserts a new user to the collection “users” using specific parameters.	email (String) password (String, plain) name (String, full name)	True
/api/v1/user/delete	DELETE	Deletes a user from the collection “users” by their email address.	email (String)	True
/api/v1/user/get	GET	Returns information about a user from the collection “users” by their email address.	email (String)	True
/api/v1/user/auth	POST	Authenticates a user using their email address and password by comparing them to information in the collection “users”.	email (String) password (String, plain)	True
/api/v1/user/checkToken	GET	Verifies if the user is authenticated by checking their Authorization Token.	token (String, **Authorization**)	True
/api/v1/test	GET	Shows internal HTML program to debug and test endpoints. Not available in public release.	-	True

Table 7: *API Server v1, 30th March 2022*

MongoDB

In order to store any type of data permanently, a certain technology that would allow doing that had to be selected. There are many database solutions such as MySQL, Cassandra or PostgreSQL to choose from, so it was decided to use MongoDB to store all kinds of data within this application. MongoDB is a document-based NoSQL database technology made up of collections that stores data in JSON- like documents.

MongoDB offers a **simple query syntax** for us to use that is easier to understand than writing SQL statements, the library is **easy to set up in languages like Python or Node.js** and has a **flexibility with data** that allows us to make changes to the structure of models without changing the scheme of a collection.
(Advantages Of MongoDB, 2022)

All layers of the project are using the same database server but rely on different collections that don't share data between each other. The API makes use of the **users** collection that stores all necessary data of individual users, such as their: email address, name, gender or date of birth. Each field is represented in a key-value format where the name of the field is set as the key and the data as the value for the entry.

Each object in the document is automatically assigned with a unique ID that is required for each document and can be used to identify the user.

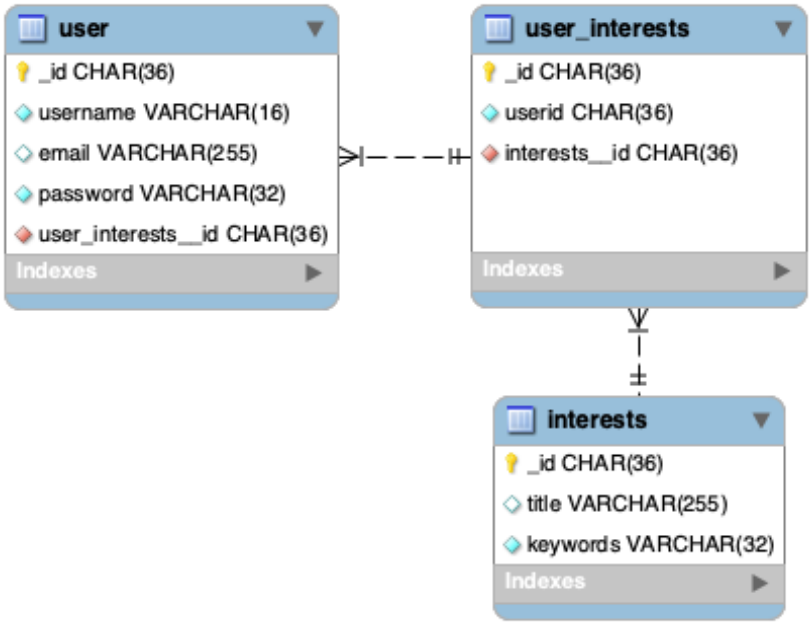


Figure 43: *Entity relationship (ER) diagram*

Jupyter Notebook

Jupyter Notebook is an open source web-based interactive computing platform. Its flexible interface allows users to arrange workflows in data science, scientific computing, and machine learning. It is an extremely powerful and, at the same time, simple tool for any type of data analysis. https://jupyter.org/

Jupyter Notebook is built off of IPython, which is a way of running Python code using the Read-Eval-Print-Loop (REPL) model. On the back-end side of Jupyter Notebook is the IPython kernel, which runs the computations and communicates with the front-end interface. https://www.codecademy.com/article/how-to-use-jupyter-notebooks

This team used this technology for some aspects of development due to its simple interface and the ability to run code in blocks, rather than running the entire script. This is useful for testing purposes, as well as it is plainly convenient.

To run Jupyter permanently, one has to run the command "(sudo) **screen jupyter notebook**" in the terminal. Once run, one has to insert their password. Then, the user is navigated to the home page (figure 32) that displays the files inside the directory, including notebooks. It is also extremely easy to upload files by clicking the upload button without bothering about FTP.

When it comes to working with notebooks, the user is given the interface shown in figure 33. It is extremely important to monitor the kernel and connection status, because, in some cases, the user can lose all of the work done after the last checkpoint, as a result of connection issues. Another common problem the user might run into is the crash of the kernel. In that case, the user has to restart the kernel and re-run the needed cells.

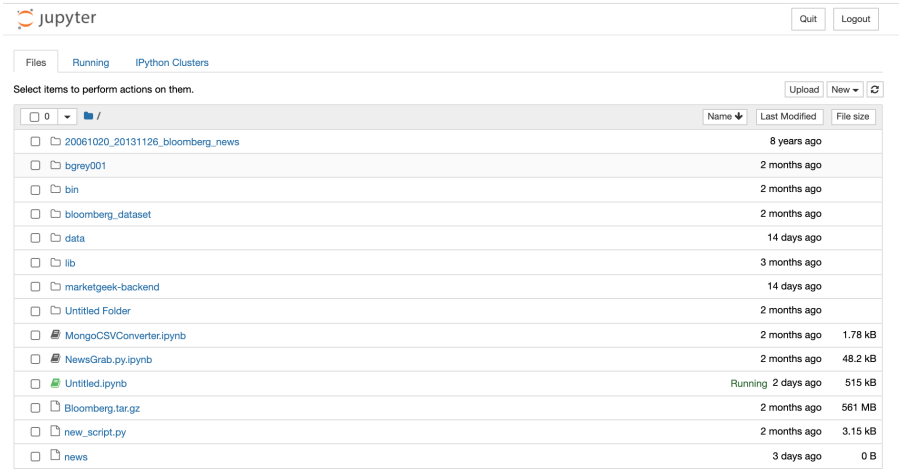


Figure 44: *view of the home page for jupyter notebook*

Figure 45: *view of the iPython notebook*

TailwindCSS

While working on this project, an external library which helps this team to fulfil the requirements for sorting tasks performed on styling React components was used.

First – this web application must have an understandable and reusable interface. Second – the functional part of this project has to be successfully implemented. Nowadays, a wide range of new technologies offers a different approach for solving UI (User Interface) problems.

In this project, it was decided to use Tailwind CSS for many obvious reasons, it can be integrated into our front-end environment which is done using React.js. Tailwind CSS has multiple advantages over other CSS libraries, such as Bootstrap.

Tailwind CSS drawbacks and advantages:

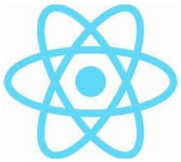
Advantages	Drawbacks
Tailwind is a CSS utility framework for styling web applications, without having a default theme that you must use like other CSS frameworks. For example, you can give each project a different look even if you use the same elements (colour palette, size, etc.).	Tailwind works differently than most CSS frameworks, and also has a declarative style of CSS rules. Tailwind mixes style rules with HTML files, which goes against the principle of the "separation of concerns." Besides that, many developers prefer to separate page structure and style, claiming that classes make the Tailwind markup process verbose.
Tailwind offers thousands of built-in classes that do not require you to create designs from scratch. Therefore, you do not have to write CSS rules yourself. These CSS classes are the main reason why building and styling with Tailwind are so fast.	Because of the built-in classes, Tailwind CSS is quite learning intensive. Even for experienced developers, it can be a challenge to learn how to use and fully utilise the pre-built classes. However, if you are confident and quick when it comes to writing CSS classes, Tailwind may not be the best choice for you. Even if that's true, Tailwind generally makes CSS styling faster in the long run.
Tailwind offers pre-built classes, for designing the layout directly in an HTML file. This makes it a very responsive, mobile-friendly CSS framework. Apart from that, Tailwind has proven to be a stable framework since its initial release. The framework was developed by top-notch engineers, which is why bugs and breaks are rare.	Unlike "Bulma" and "Bootstrap", Tailwind does not have many significant styling components. Unfortunately, this means you must manually add features like headers, buttons, and navigation bars for web apps.

Tailwind CSS allows users to write utility classes instead of using predefined components whilst taking away vanilla CSS syntax that you won't use anymore. Instead of relying on pure CSS, Tailwind has an option to add short keywords of CSS functions within the HTML elements. An example would be "ml-4" (translates to margin-left: 4px) or "md:text-left" (translates to text-align left; for medium screen sizes).	Although Tailwind CSS has made great strides when it comes to adding guides and video tutorials, it is still behind competitors like Bootstrap. Of course, you can always contact the developers if you have a problem.
Tailwind also handles responsive web design using parameters like "sm", "md" or "lg" for different screen sizes or even dark and print versions of the website with "dark" and "print". The utility classes only apply to the respective settings then, so "dark:text-white" would only make the text white when the user's OS is set to the dark mode or "print:text-2xl" would set the text size bigger when the user prints your web page.	-

Table 8: *drawbacks and advantages of TailwindCSS*

After careful consideration and comparison of all advantages and drawbacks of potential candidates for CSS frameworks in this project, a conclusion was made that Tailwind CSS completely fulfils this project's goals and requirements. ("THE PROS AND CONS OF TAILWIND CSS", n.d.)

React.js



React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex User Interfaces from small and isolated pieces of code (“components”). (“What Is React?”, n.d.)

Setting up a React.js project and integrating Tailwind CSS

This team commences the front-end part of this project by installing all necessary Node.js libraries (Latest LTS Version: **16.14.1** (includes npm 8.5.0), available as open source on (“Download the Node.js source code or a pre-built installer for your platform”, n.d.).

In order to use TailwindCSS in our React.js project, it must first be integrated. The Tailwind CSS library provides a command to set up a new React.js project that is not much different to the original “create-react-app” command you would use when setting up a React.js project.

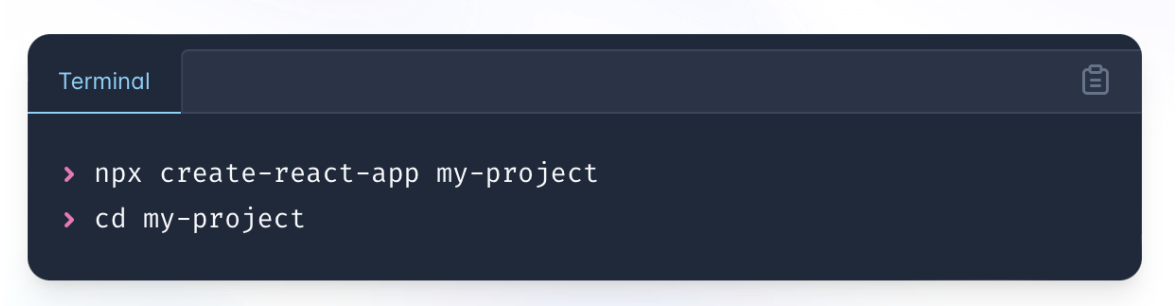


Figure 46: *Setting up React project*

After creating a project folder on the local machine, using the command Prompt on Windows and Terminal on macOS **npm start** (node package management) was performed, which is a predefined command to start the development server using React.js. This command also runs the website on all widely recognised browsers using the address “http://localhost:3000”.

React almost simultaneously renders the project on a web browser; it takes a fraction of a second to compile the code. In case of failure, “localhost” can be rendered manually, just adding port “3000” to the URL section of the browser. This port is used by default for the React development server. (“Creating a React Project”, Justin Lange, 2022)

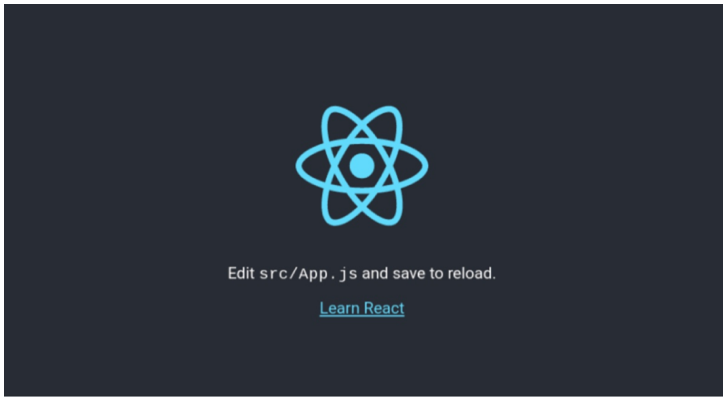


Figure 47: *React New Project Screen*
You should be able to see this screen (or similar) in your browser

The commands automatically initialise a Git repository in the directory of the project. This makes the use of Git a lot easier and does not require us to initialise a repository.

The npx command automatically creates a package.json. In any other case, a package.js file can be manually created using the npm-init command. (“npm-init Create a package.json file”, n.d.)

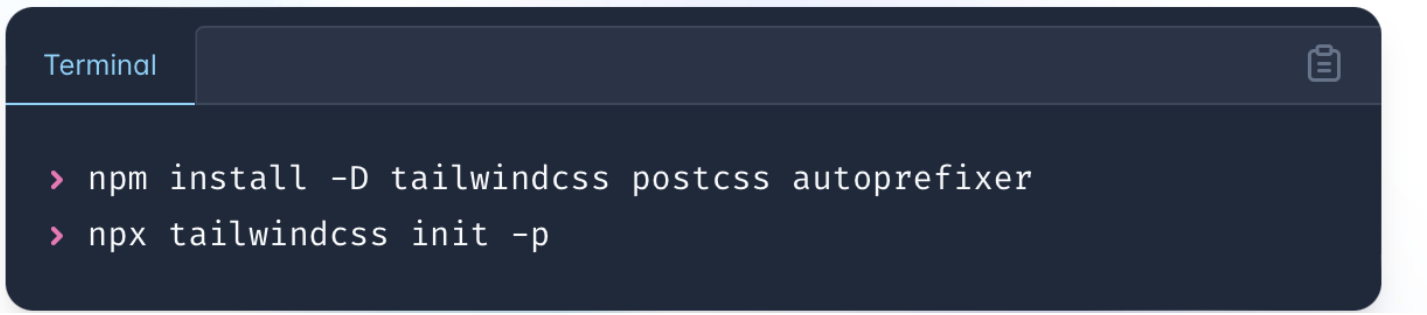


Figure 48: *Setting up React project*

In order to use Tailwind CSS in the created React.js project, a few required packages need to be installed using the built-in npm package manager. These include TailwindCSS itself**, postcss** as well as **autoprefixer.** The packages will be installed in a “node_modules” folder. Due to the increasing size of the folder, this one should not be pushed onto a Git repository and be included in a **.gitignore** file to prevent this from happening.

PostCSS is a tool that’s making the rounds on the front-end side of web development. It is a Node.js package developed as a tool to transform all used CSS using JavaScript, thereby achieving much faster build times than other processors. (“PostCSS: Sass’s New Play Date”, n.d.)

Autoprefixer is a **PostCSS** plugin to parse CSS and add vendor prefixes to CSS rules using different values. It also allows us to make our own CSS rules without prefixes.

After installing and initialising the Tailwind CSS package, it should automatically create a tailwind.config.js. In addition to the predefined configuration, it is required to also add any files where TailwindCSS will be used.

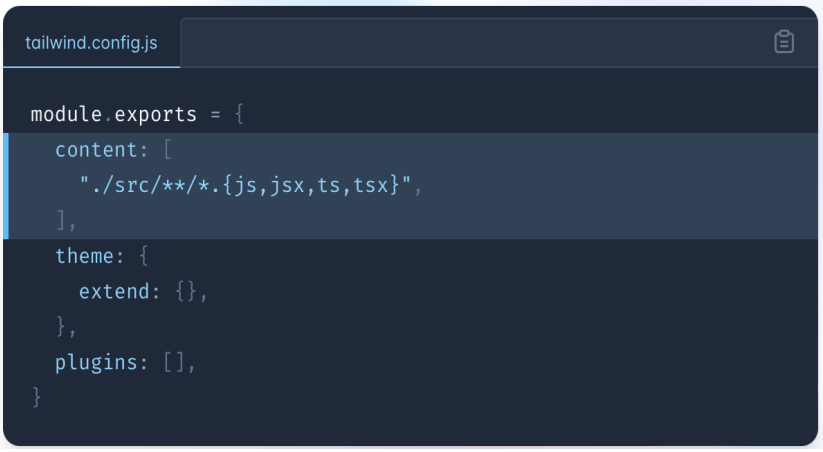


Figure 49: *Tailwind Configuration file in the project root directory*

The parser will be going through the files and adding the used CSS classes (e.g. “text-centre” or “text-2xl”) to the final CSS file which will be used in a public build of the web application.

index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Figure 50: *adding the Tailwind directives to CSS*

Tailwind CSS consists of three main packages: base, components and utilities. All three packages are required to use TailwindCSS in its complete form. The packages need to be declared in a CSS file which is then imported in the Main App View (**App.jsx**).

App.js

```
export default function App() {
  return (
    <h1 className="text-3xl font-bold underline">
      Hello world!
    </h1>
  )
}
```

Figure 51: *Initialising a new project in Tailwind*

The library and React.js project is set up now and the front end team is able to begin the work on migrating proposed designs into a prototype. The development server which is built into React.js can be started using the **npm start** command.

Search page and functionality

The search panel has been represented in the project as a separate page with the ability to work on an end-user desktop machine, smartphone, or tablet PC. Search page implemented with accessibility features working in dark mode and light mode. These features have been implemented using React and Tailwind CSS in the Search.jsx file.

The main function of the search panel is to perform user requests, searching for topics of user interest. The searching panel query was implemented using a SearchResult.json file that contains all necessary information generated by the back-end-side. **Figure 49** represents the UML class diagram which describes the interrelation between Search.jsx class components and API represented as formatted JSON in this application.

Search Results - it's a searchResult.json file which stores all necessary information for a successful target of interest search. The user would not get redirected to any other page; all necessary information will get rendered on the same search page. The search page contains a functional search bar that helps users enter and then search for information of interest. The search bar functionality is responsible for the SearchBox.jsx component, specifically the onChange event handler. Vanilla JavaScript has an attribute onchange, React's version is the same, but camel-cased. An onChange event handler returns a [1] Synthetic Event object which contains useful metadata such as the target input's id, name, and current value.

Company / Stock Information – used in the MarketGit web application is the external "Polygon" API where the team has sent requests to and received the required information about the top result as a JSON output. Following attributes are included in the response:

---

[1] Event handlers will be passed instances of SyntheticEvent, a cross-browser wrapper around the browser's native event. It has the same interface as the browser's native event, including stopPropagation() and preventDefault(), except the events work identically across all browsers. ("SyntheticEvent", n.d.)

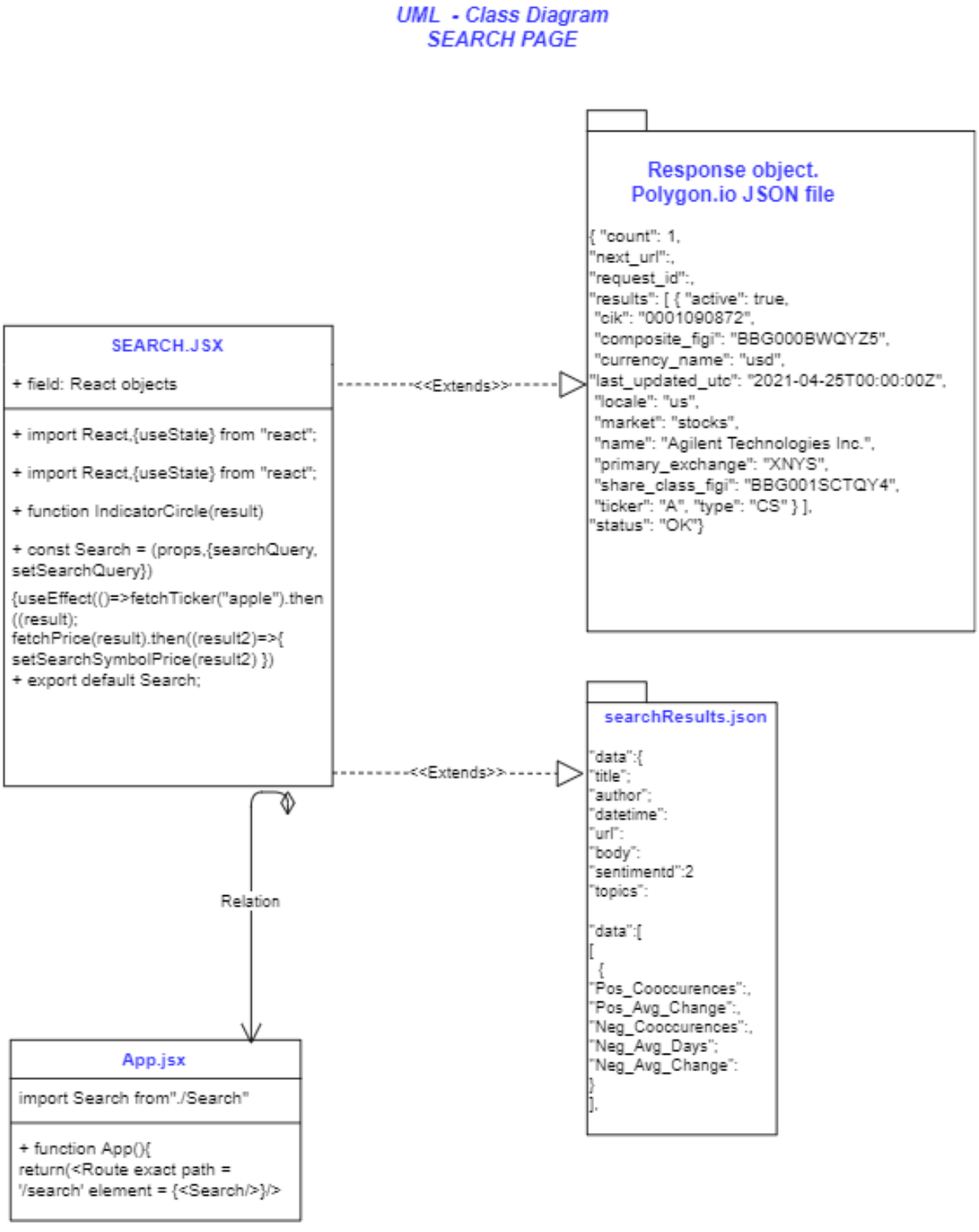


Figure 52: *UML class diagram for the search page*

Response Attributes

Attributes	Description
Results array	An array of tickers that match your query.
active boolean	Whether or not the asset is actively traded. False means the asset has been delisted.
cik string	The CIK number for this ticker.
composite_figi string	The composite OpenFIGI number for this ticker.
currency_name string	The name of the currency that this asset is traded with.
last_updated_utc string	The information is accurate up to this time.
delisted_utc string	The last date that the asset was traded.
locale *enum [us, global]	The locale of the asset.
name *string	The name of the asset. For stocks/equities this will be the company's registered name. For crypto/fx this will be the name of the currency or coin pair.
ticker *string	The exchange symbol that this item is traded under.
type string	The type of the asset.
status string	The status of this request's response.

Table 9: *Polygon API response attributes*

Evaluation

Back-end

Conceptual issues

Data issue: time frames for closing prices

The idea that trending news articles influence price trends of certain financial assets assumes that most of those influences exert their effects mostly in a short term setting.For instance, after Facebook released its quarterly figures in February this year, share price fell by 26.4% in a single day (“Facebook suffers $230bn wipeout in biggest one-day US stock plunge” 2022) . While long term effects of trending news on certain assets are less apparent.

Since yfinance library (Aroussi, n.d.) that uses public Yahoo Finance APIs was the primary choice for gathering financial data, it is not possible to acquire a 15 minute interval price history of certain financial assets, since this API only allows 1-day interval for data older than one year before the current date.
This is a downside in terms of this projects’ needs, however, it was still possible to demonstrate the fundamental concept this team proposed.

Optimal solution for topic generation

Using something like keyword matching, where one has a dictionary containing categories as keys and lists of terms that belong to those categories as values, was an alternative possibility (“Assigning categories to tweets using keyword matching”, n.d.). This would result in more clear and representative topics, since one doesn’t have to deal with complexities of tuning various machine learning models, however, this approach would require a lot of time for creating a database with categories and terms that are associated with it. This could be considered an enterprise level task, but in the scenario of an extremely limited timeframe, we decided to resort to automatic topic generation not to lose momentum.

Importance of time delta

Besides parameters such as the number of topics, or minimum number of samples that are crucial for producing consistent results, one of the most important parameters for the overall algorithm that gets the co-occurrences is the time delta. Time delta was used as a parameter for gathering news in a certain time frame. For instance, in the current implementation, time frames from two days until 12 hours before the price trend initiation were considered. This was due to the inherent lag that the EMA crossover strategy possesses. As outlined in this post (“Exponential Moving Average - Lag” 2008), lag for regular EMA is the window span divided by two, and for double

EMA it is the window span divided by four. Considering this, it is crucial to analyse the correct time frame, as such, parameters like window spans for the EMA and time delta have to be selected accordingly.

Downside of moving averages

Using EMA does not always give consistent results because of false breakouts, which are predominant during the episodes of high volatility. That means that sometimes the trends that are registered are false. As such, the resulting outputs of EMA crossover strategy implementation might not show consistent results i.e. upward trends with the price at the start being higher than the price at the end of the trend, and vice versa (Schmitt, n.d.).

Further ideas to make this product fully functional

Ideally, to make this a fully functional product, one would require a full dataset with no gaps in dates up until the current time. If this team could find a dataset that fills the aforementioned requirements, the output results would largely improve. BERTopic's pipeline could be still used to cluster the articles with a fixed number of topics, but the actual output of generated topics that are shown to the user is going to come from another technique. The best algorithm to do such a thing would be just a simple keyword matching ("Assigning categories to tweets using keyword matching", n.d.) as mentioned above.

Another idea would be that the front end interface would have a button for sorting in order of negative co-occurrences and positive co-occurrences, or even other data the co-occurrence algorithm generates. But for this demo version only a sorting option for positive co-occurrences is available.

News dataset: steps that were taken and alternative options

As with stock data, gathering a financial news dataset was largely cumbersome. One option was to scrape news articles from the web, however, being mindful of the deadline, this team decided to collect a complete dataset, which only requires preprocessing. The search for the proper dataset was also complicated. At first, Reuters was considered as the primary source of a news collection. But some of the content this resource provided was not relevant to the scope of this project. Another option was a dataset from kaggle, which was a combination of financial news of over 6000 stocks ("Stock news analysis", n.d.). However, selecting this dataset was essentially limiting, as only stocks would be considered, while excluding currencies, commodities, or indexes, which are more interesting to analyse and find relationships with.

As such, it was decided to employ a Bloomberg financial (Remy and Ding, n.d.) news dataset from 2006 to 2013 with a total of over 450,000 entries.

Future back-end implementations

Due to time constraints on development there are features that were not able to be fully implemented. The most important of these implementations included the automatic retrieval of news articles from multiple sources

across the UK and USA to be run through the analysis models. The retrieval used requests based on the news-api (“News API”, n.d.) along with tools from pandas to grab data using set payload parameters depending on the region and moved into a dataframe using pandas along with saving a local csv onto the server. Figures (53-55). The final part of the implementation, Figures 54 allows for data to be sent to an API server depending on the title request.With these implementations in place the gap between the back-end and frontend would have a theoretical bridge to allow for continuous updates.

```
from newsapi.newsapi_client import NewsApiClient
import requests
import json
import pandas
from io import StringIO


from flask import Flask, request

newsapi = NewsApiClient(api_key='[redacted]')

#grab headlines from news-api
top_headlines_url = 'https://newsapi.org/v2/top-headlines'
everything_news_url = 'https://newsapi.org/v2/everything'
sources_url = 'https://newsapi.org/v2/sources'

#dictionary
struct_dict = {'title': [], 'author': [], 'datetime': [], 'source': [], 'body': []}
df = pandas.DataFrame(struct_dict)

headers = {'Authorization': 'ee1453e25ade4e16b8fb219f9fe9e6f5'}

#set parameter for request
headlines_payload__business_usa = {'category': 'business', 'language': 'en','country': 'us'}
headlines_payload_business_uk = {'category': 'business', 'language': 'en','country': 'gb'}
everything_payload = {'q': 'business', 'language': 'en', 'sortBy': 'popularity'}
```

```
#grab the headlines from locations specified in parameters
response_uk = requests.get(url=top_headlines_url, headers=headers, params=headlines_payload_business_
response_usa = requests.get(url=top_headlines_url, headers=headers, params=headlines_payload__busines

#convert the responce into json so can be output as a string
output = json.dumps(response_uk.json(), indent=2)
output2 = json.dumps(response_usa.json(), indent=2)

#print(output,output2)

#turn strings into a array so we can seperate the articles
responses_uk = json.loads(output)
articles_list_uk = responses_uk['articles']
output_articles_uk = pandas.read_json(StringIO(json.dumps(articles_list_uk)))

responses_usa = json.loads(output2)
articles_list_usa = responses_usa['articles']
output_articles_usa = pandas.read_json(StringIO(json.dumps(articles_list_usa)))

output_articles = output_articles_uk.append(output_articles_usa).reset_index(drop=True)
```

Figure 54: *C1-P2: Python NewsApi Grabber*

```
#adding values from the output articles data (stored as objects within array) to a larger dataframe t
df["title"] = output_articles["title"]

df["author"] = output_articles["author"]

df["datetime"] = output_articles["publishedAt"]
df["source"] = output_articles["url"]
df["body"] = output_articles["description"]

#write the outputs into a csv
df.to_csv('newsout.csv')
```

Figure 55: *C1-P3: Python NewsApi Grabber*

```python
#df.query('title == "Stocks stumble as Ukraine tensions worsen, investors seek safety in bonds - Reut
result = df.to_json(orient="split")

# Initalise the backend web app
app = Flask(__name__)
api_version = "/v1"

# Routes for the API

# Route to filter by title
# Example: http://127.0.0.1:5000/v1/news/getByTitle?title=Take it or leave it: second hand picks for
@app.route(api_version + "/news/getByTitle", methods=['GET'])
def get_data():
    # Query the result by title
    df.query('`title` == "' + request.args.get("title") + '"', inplace = True)

    # Convert to JSON
    json_dict = df.assign(index=df.index).to_dict(orient="list")

    # Return the result
    return json.dumps(json_dict)

# Run the API server
if __name__ == "__main__":
    app.run(debug=True)
```

Figure 56: *C1-P4: Python NewsApi Grabber*

Technical issues

Case 1: **Cleaning and merging the dataset**

One of the first major challenges on the back end of the application was transforming the large financial news dataset into a format that the team could effectively read and access. The initial dataset consisted of a large number of directories with subdirectories with unnamed files in each one that represented a single news article. The articles themselves were in a highly unstructured format that had many unwanted symbols, spaces and characters that were not important for the use case. The problem was solved by writing scripts to walk through the subdirectories and read each file, simultaneously cleaning and compartmentalising the data into a pandas dataframe that we exported to a comma separated values file (csv).

```
def walk_files(filepath):
    for subdir, dirs, files in sorted(os.walk(filepath)):
        i = 0
        for file in sorted(files):
            if i < 2:
                path = os.path.join(subdir, file)
                if get_structure(path) is None:
                    continue
                else:
                    dfs.append(pd.DataFrame(get_structure(path)))
                i += 1
```

***Figure 57**: parsing the text from numerous text files*

```
def get_structure(filepath):
    file_text = ""
    try:
        struct_dict = {'title': '', 'author': '', 'datetime': '', 'source': '', 'body': ''}
        with open(filepath, 'r') as f:
            for idx, line in enumerate(f):
                line = re.sub("\n", " ", line)
                file_text += line
            file_text = file_text.replace('--', '&')
            file_text = file_text.replace('.html', '&')
            file_text = file_text.lower()
            file_text = file_text.encode('ascii', 'ignore').decode()

            array = file_text.split('&')
            struct_dict["title"] = [array[1].strip()]
            author_array = [array[2].strip()][0]

            if author_array.find('b y ') != -1:
                author_array = re.sub('(?<!\s)\s{1}(?!\s)', '', author_array)
                author_array = re.sub('\s\s+', ' ', author_array)
                author_array = author_array.replace('by ', '')

            struct_dict["author"] = [author_array]
            struct_dict["datetime"] = [array[3].strip()]
            struct_dict["source"] = [array[4].strip() + '.html']
            struct_dict["body"] = [array[5].strip()]
    except:
        return
    return struct_dict
```

***Figure 58**: cleaning and formatting strings parsed by figure 46 (above)*

Case 2: **Comparing news**

The team needed to find the best way to compare news, so, by utilising the BERT model (see BERT in research), the team embedded the news titles and performed HDBSCAN clustering. Now, this representation of similar news (news that belongs to the same topic cluster), is very convenient for the future goals of this project.

Case 3: **Improving computation time**

In terms of computational resources, the team had allocated an API server to the project. However, there were serious limits on the power of the allocated Microsoft Azure server. As a result the back-end team was spending a significant amount of time waiting for the computation of certain tasks. This led to a decision to use one of the team members' local computers which had an up to date Ryzen 5950x CPU and NVIDIA RTX 3090 GPU, allowing for a much faster computation time when optimised with the CUDA toolkit (see in planning/research). The process enabled by CUDA cut an initial estimate from 8 hours down to 20 minutes.

```
14    cuda = torch.device('cuda')
```

***Figure 59**: using CUDA with pytorch (Python)*

Case 4: **Simplifying the implementation of the BERTopic model**
Once run on a GPU, fine tuning the model was relatively fast. However, iterating through each news article and transforming took a vast amount of time. Once again, we had to find a new approach. This was solved by transforming the whole dataset as one, outputting a single list of vectors that contained the topic id of each respective news article. These were then mapped to the original dataset.

Case 9: **Setting the parameters of the model**
Choosing the number of clusters for sentiment and topics. It is problematic, since there are no definitive heuristics on how to select the number of clusters for a particular model. The team has chosen 150 clusters for topics, and 15 topics for sentiments. Since classic hierarchical density-based algorithms initially have no parameter that defines the number of clusters, the team encountered cluster numbers that are in the thousands, which would not be optimal for comparing similar news, since there is a smaller probability of getting matches. Another issue with selecting the default BERTopic parameters was that almost a half of the output contained outliers, which can happen in case of a noisy dataset, or if the parameter min_cluster_size is set to higher numbers. Min_cluster_size is a parameter which defines how many data points are enough for them to be considered a cluster.

Case 5: **Reducing the number of topic clusters**
The model returned too many clusters, which would be a problem when we start identifying similar news at different time stamps for displaying the statistics to the user. This would result in a non-comprehensive output for our purpose. Tried solving it by playing around with the number of topics, setting it to values from 50 to 180.

When implementing BERTopic modelling, Python was used to write scripts that took training data to fine tune the BERTopic model to a given dataset (using 80% of the dataset as suggested here {find link}). Following this, we used the full dataset of news articles to generate the topic clusters. The team appended the topic id with the list of words in that associated topic into the dataset.

The team tested multiple variations of hyper parameters for BERTopic, using an iterative testing approach to fine tune the results. After our initial testing on small sample data we were satisfied with the results. However, when run on the full dataset we realised we had far too many outliers (in this context this means articles that were in a topic cluster that was very common and therefore had no value). As a result, we had to tweak the minimum topic size. This yielded satisfactory results.

```
76    topic_model = BERTopic(min_topic_size=100,
77                           calculate_probabilities=False,
78                           low_memory=True,
79                           verbose=True,
80                           n_gram_range=(1, 2))
```

***Figure 60**: the final parameters of the BERTopic model (Python)*

Case 6: **Including sentiment**

By reducing the topics, we might assign news that are of different sentiments to the same cluster, which should be avoided, since once we start comparing news in the beginning of positive and negative price trends - news sentiment is what is going to define the distinction, not the semantics of the sentence. To compute the sentiment of each news article we used FinBERT (as outlined in the planning section).

```
tokenizer = AutoTokenizer.from_pretrained("ProsusAI/finbert")
model = AutoModelForSequenceClassification.from_pretrained("ProsusAI/finbert")
inputs = tokenizer(sentences, padding = True, truncation = True, return_tensors='pt')
outputs = model(**inputs)
predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
predictions = predictions.detach().numpy()
```

***Figure 61**: Implementing FinBERT and outputting the predictions as tensors (Python)*

Case 8: **Batching due to memory errors**

When processing data, transforming and fitting models, the team repeatedly came upon memory allocation errors that were killing the process. This termination occurred at various different times depending on the script that was running. This posed a major problem as the time taken to run certain scripts for topic clustering took over an hour and a half, at times this would terminate after over 50% of the process had been completed, wasting a great deal of time. A solution the team had to learn and then deploy was batching. Batching is the technique of breaking up a dataset and the operations applied to the data into smaller chunks, allowing the memory of the computer to be released and reused for the next chunk. This was employed for all subsequent large data operations.

```
for chunk in tqdm(np.array_split(df, 1024)): #1024 split
    titles = list(chunk['title'])
    inputs = tokenizer(titles, padding = True, truncation = True, return_tensors='pt')
    outputs = model(**inputs)
    predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
    predictions = predictions.detach().numpy()
    tensors += list(predictions)
```

***Figure 62**: Batching sentiment using FinBERT model (Python)*

Case 10: **Tensor computation**

The team ran into an issue while producing the tensors for sentiment cluster ID's. The process was computationally expensive, taking about 90 minutes in total. Despite running tests using smaller sample sizes, we still had to make adjustments to our scripts to avoid recursion errors when executing on the whole dataset. After researching and reviewing the problem we identified the possible fault - performing clustering on a large dataset and not adjusting sample size to account for this.

Case 12: **Constraining sentiment IDs to three values iteratively**

After experimenting with various values in the parameters for HDBSCAN we decided that the results were too broad. In addition to this, there were numerous outliers which we felt were unnecessary for basic sentiment analysis. As a result, we decided to take a more simplistic yet logical approach. This entailed taking the greatest value from each tensor and outputting the id based on the index of this max value. This gave three possible results for positive, negative and neutral which correspond to 0, 1 and 2 respectively. It is important to note that this does not account for the true diversity of the range of sentiments. However, this team concluded that this approach will be effective in combination with topic analysis when producing the co-occurrences.

```python
tensors = np.load('data/tensors.npy')
max_indexes = []

for tensor in tensors:
    max = 0
    for t in range(0, 3):
        if tensor[t] > max:
            max = tensor[t]
            max_index = t
    max_indexes.append(max_index)

df = pd.read_csv('data/topic_size_min_100.csv')
df['sentiment_id'] = max_indexes
df.to_csv('data/full_dataset_march15.csv', index=False)
```

***Figure 63**: Finding the dominant float value in the tensor to determine the sentiment (Python)*

Case 13: **Converting strings to datetime**

In order to find co-occurrences, strings had to be converted from the MongoDB tickers and news to Python datetime format. The strings from each respective collection were structured differently. As a result they had to be parsed into the datetime format using the datetime library in Python.

```python
upwards_ticker = pd.read_csv('data/tickers/EURUSD=X_upward_trends.csv')
downwards_ticker = pd.read_csv('data/tickers/EURUSD=X_downward_trends.csv')
date_string = str(upwards_ticker['Start_Date'].iloc[len(upwards_ticker) - 1])
datetime_object_ticker = datetime.strptime(date_string, "%Y-%m-%d %H:%M:%S")
prev_day = datetime_object_ticker - timedelta(days=2)
df['datetime'] = pd.to_datetime(df['datetime'], format="%Y-%m-%dt%H:%M:%Sz")
df.to_csv('data/full_dataset_march16.csv', index=False)
```

***Figure 64**: Converting string to datetime (Python)*

Case 14: **Retrieving trends for financial assets**

It was necessary to first find all of the trends for every single financial asset that was considered in this project. As such, the ticker in question was read from the database and analysed in the manner as seen in Figure x.

```python
def trends_for_ticker(ticker, short_term = True):


    coll = financedb[ticker]
    df_prices = pd.DataFrame(list(coll.find()))

    if short_term:
        df_prices["DEMA_12"] = dema(df_prices, 12, 'Close')
        df_prices["DEMA_26"] = dema(df_prices, 26, 'Close')
        df_prices['Signal'] = 0.0
        df_prices['Signal'] = np.where(df_prices['DEMA_12'] > df_prices['DEMA_26'], 1.0, 0.0)

    else:
        df_prices["DEMA_50"] = df_prices["Close"].ewm(span = 50, min_periods = 1).mean()
        df_prices["DEMA_200"] = df_prices["Close"].ewm(span = 200, min_periods = 1).mean()
        df_prices['Signal'] = 0.0
        df_prices['Signal'] = np.where(df_prices['DEMA_50'] > df_prices['DEMA_200'], 1.0, 0.0)

    df_prices['Position'] = df_prices['Signal'].diff()
    trends = df_prices[np.logical_or(df_prices['Position'] == 1, df_prices['Position'] == -1)]

    trends['Start_Date'] = trends['Date']
    trends['End_Date'] = trends['Start_Date'].shift(-1)
    trends['Close_Start'] = trends['Close']
    trends['Close_End'] = trends['Close'].shift(-1)

    trends = trends[:-1]

    trends = trends.drop(['_id', 'Low', 'High',
                          'Close', 'Date', 'Adj Close',
                          'DEMA_12', 'DEMA_26', 'Signal',
                          'Volume', 'Open'], axis=1)

    trends.loc[(trends['Close_Start'] > trends['Close_End']), 'Trend'] = -1
    trends.loc[(trends['Close_Start'] < trends['Close_End']), 'Trend'] = 1
    trends[['Start_Date', 'End_Date']] = trends[['Start_Date', 'End_Date']].apply(pd.to_datetime)

    return trends
```

Start_Date	End_Date	Close_Start	Close_End	Trend
2006-10-25	2006-12-14	1.261607	1.315651	1.0
2006-12-14	2007-02-01	1.315651	1.301897	-1.0
2007-02-01	2007-03-09	1.301897	1.311596	1.0
2007-03-09	2007-03-12	1.311596	1.318791	1.0
2007-03-12	2007-05-02	1.318791	1.358806	1.0
...	...	...	...	...
2013-07-17	2013-08-29	1.315115	1.333493	1.0
2013-08-29	2013-09-17	1.333493	1.333796	1.0
2013-09-17	2013-10-16	1.333796	1.352174	1.0
2013-10-16	2013-10-21	1.352174	1.368195	1.0
2013-10-21	2013-11-01	1.368195	1.357976	-1.0

Figure 65 (left):** Function for retrieving price trends for each ticker (Jupyter Notebook), **Figure 66 (right):** Output of the function in **Figure 65.

Case 15: **Resolving trend inconsistencies**
In order to get co-occurrences only buy and sell signals were considered. The approach that was used to find beginnings and ends of price trends was not precise, due to the issue outlined above. This team managed to overcome this issue by comparing the respective close prices, leading to an accurate indicator for the price direction.

Case 16: **SettingWithCopyWarning in Python**
SettingWithCopyWarning is not an error but a warning. The warning comes about due to the nature of working with pandas dataframes. Creating a subset of a dataframe in pandas can be done in two ways, either through a view or a copy. A view is still working with the original dataframe and therefore any changes made modify the original. However, if a copy is made it is independent, any changes will only impact the new subset and not the original. The team encountered this warning multiple times during our development; however it was a false positive. This was due to the fact we were not concerned with creating and editing copies as we were appending them to separate objects independently of the original pandas dataframe view.

```
<ipython-input-20-2e041b0b7f15>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
pd.options.mode.chained_assignment = None
```

***Figure 67**: example of SettingWithCopyWarning (above), solution to the problem (below) (Jupyter Notebook)*

Case 17: **Exporting JSON to frontend**
Exporting as JSON for front-end. After developing the functionality of the application that outputs co-occurrences based on the input of a single news article, we had to transfer these results to the front end team. This was achieved by selecting and formatting the appropriate elements of data. After an iterative testing process with the front end team, we adjusted the output data to include the positive and negative co-occurrences, positive and negative average price changes and positive and negative average number of days, for every financial asset in the dataset. The team exported five news articles selected at random with their respective data as a JSON, as requested by the front end team.

```
In [471]: news_articles = [df_news.iloc[255000], df_news.iloc[322166],
                          df_news.iloc[324166], df_news.iloc[325156], df_news.iloc[360156]]
          f = open('news.json', 'w')
          for news in tqdm(news_articles):
              data = get_tickers_json(news, df_news)
              f.write(data)
              f.write("\n")
          f.close()

          100%|██████████| 5/5 [00:23<00:00,  4.64s/it]
```

***Figure 68:** Exporting the JSON (Jupyter Notebook)*

Case 18: **Improving accuracy of the trends**
Algorithm for predicting trends was producing inconsistent results. This was due to only considering buy or sell signals, not both combined. The team amended this and can now predict the start and end of trends accurately. Now we have accurate trends we can take the average price change.

Case 19: **Storing data**

Storing the news dataset as a CSV (comma separated values) format was not the optimal form of storage for some of the data types. This was due to the diverse range of types in the dataset. Repeatedly, when exporting as a CSV from a pandas or dask dataframe, certain data types would be converted to a string. This was not suitable. Integers and floats were preserved; however, tensors and lists of topics were converted into strings. As such, the team had to respectively use a separate numpy file to store tensors and parse a list of topics from a string back to a list using regular expressions.

```
34    np.save('data/tensors.npy', tensors)
```

***Figure 69**: saving tensors as a numpy file to preserve type*

Case 20: **Getting the co-occurrences**

The process of obtaining co-occurrences is relatively straightforward. First of all, the data frame of a ticker in question, the historical financial news dataset, and the news article that is being analysed have to be provided. Then, after initialising the variables needed to help build the output dictionary, the program iterates through every row in the provided ticker data frame. Inside the loop, the news articles that are within a certain time frame (minus 2 days up to minus 12 hours) before the start date of the price trend and are related to the query news article are retrieved. After this step, the if conditional checks if the trend is upwards and downwards and adds the necessary data to the variables that will compute the average once the loop ends. Once the average is computed, the final values are inserted into the output dictionary. In this implementation, the use of dictionaries stems from the convenience of storing dictionaries as a JSON file, using the json.dumps() function.

```
In [404]: def get_all_sim_news(news, df_news, df_tickers):

              output_dict = {}

              pos_length = 0
              pos_days = timedelta()
              pos_price = 0

              neg_length = 0
              neg_days = timedelta()
              neg_price = 0

              for row in df_tickers.iterrows():
                  news_in_tf = get_news_in_timeframe(df_news, row[1]["Start_Date"])
                  sentiment_id = news['sentiment_id']
                  topic_id = news['topic_id']
                  if topic_id != -1 :
                      sim_news_in_tf = news_in_tf.loc[
                      np.logical_and(news_in_tf['sentiment_id'] == sentiment_id,
                                     news_in_tf['topic_id'] == topic_id)]

                      sim_news_in_tf = sim_news_in_tf.drop_duplicates(subset=['query_date'], keep = 'last')

                      sim_news_in_tf['Trend_Start'], sim_news_in_tf['Trend_End'] = row[1]['Start_Date'], row[1]['End_Date']
                      sim_news_in_tf['Closing_Start'], sim_news_in_tf['Closing_End'] = row[1]['Close_Start'], row[1]['Close_End']

                      if row[1]["Trend"] == 1 and len(sim_news_in_tf) >= 1:
                          pos_length += len(sim_news_in_tf)
                          pos_days = pos_days + (row[1]['End_Date'] - row[1]['Start_Date'])
                          pos_price = pos_price + ((row[1]['Close_End'] - row[1]['Close_Start'])/row[1]['Close_Start'])*100

                      elif row[1]["Trend"] == -1 and len(sim_news_in_tf) >= 1:
                          neg_length += len(sim_news_in_tf)
                          neg_days = neg_days + row[1]['End_Date'] - row[1]['Start_Date']
                          neg_price = neg_price + ((row[1]['Close_End'] - row[1]['Close_Start'])/row[1]['Close_Start'])*100


              output_dict["Pos_Cooccurences"] = pos_length
              output_dict["Pos_Avg_Days"] = str(pos_days/pos_length)
              output_dict["Pos_Avg_Change"] = pos_price/pos_length

              output_dict["Neg_Cooccurences"] = neg_length
              output_dict["Neg_Avg_Days"] = str(neg_days/neg_length)
              output_dict["Neg_Avg_Change"] = neg_price/neg_length

              return output_dict
```

***Figure 70:** a snippet of code that obtains the co-occurrences. (Jupyter Notebook)*

Front-end

The front end application was built using different libraries that helped us throughout the development of the proposed ideas and features. The integration of the design has begun by taking a deeper, closer look at the previous design mock-ups and deciding how the best possible first user impression can be created. This team knew that the first impression of the website could decide if a client decides to create an account, so we wanted to make sure that we keep the design clean, use a specific range of colours as shown before in section Planning and Design and Development as well as integrating a clear headline to the user that explains them what we offer.

During a session of brainstorming, this team decided to come up with a short headline, two buttons to avoid the user from clicking through too many pages but ensure that the user can find out more about the features by scrolling further down on the home page. Instead of redirecting the user to different pages for creating a new account or signing in, this team decided to show modals and give users the opportunity to close them as natively to their existing OS experience as possible. The idea of simplicity helped this team to make this decision, and user testing as shown in {} has confirmed this approach.

One of the first challenges that were faced during the development of the frontend application was the collaboration between team members. This team thought the idea of contributing to one main branch on GitHub would be too confusing and might cause issues when making bigger changes to the code base, so instead we created branches for each member and later merged the changes together. This helped massively during code-reviews when the team compared new code and checked for aspects that could be improved.

Technical issues

Case 1: **New APIs in React**
Since the newest React version, this team faced many problems when implementing certain features into this website. This happened at the beginning when this team tried to create new routes within our application to separate views from each other using URLs (such as **/login**, **/home** or **/terms**). The APIs have drastically changed over the time, so this team had to adapt to the changes and couldn't rely on most resources found online. The decision was made to stick to the newest React version for security, features and adoption rate. (INCLUDE PROOF)

Case 2: **Separation of components**
During the development of the index page, this team noticed that the JSX code became quite long and hard to read with the different HTML elements. To prevent this from slowing down this project's development process in the future, it was decided to create a separate folder that stores components (such as Buttons, Grid System, Text fields) in different JSX files. These components were exported and this team had no problem to reuse them in all of the views.

Case 3: **Exposure of the API Key**
After testing a first prototype build of the web application, this team noticed that the API key which was defined in the .env file of the React project is readable for any user in the JavaScript source code and HTTP requests log. Due to the time constraints, the team was not able to fully solve this, but solutions such as restricting incoming

requests to the API Server for certain IP addresses and moving the React project to a server-side solution such as using Node.js or a similar framework Next.js were proposed.

Case 4: **Receiving information by date**
Implementing a widget with the market's latest news update, the front-end team faced several technical challenges related to correct data rendering. For the latest news section implementation, Polygon.io Stocks API was used to gather the external data.

These APIs provide REST endpoints that let developers query through the latest market data from all US stock exchanges. It is also able to find data on companies' financials, stock market holidays, corporate actions, and more. Polygon REST APIs are based on entitlements that control which endpoints you can use and which kinds of data you can access. All users start with free access to reference data and end-of day market data. (Vats, n.d.)

Because of this, the team decided to have a free data subscription, there were no other choices, but only access to yesterday's day stock market closing data. The challenge that we faced with requesting this data, was a real-time format data request, which had a contradiction with the subscription that we had chosen. To avoid this issue, the function getYesterdayDate() **Figure 71** was created; this function was successfully used for fetching yesterday's data from the Polygon.io web resource.

```
function getYesterdayDate() {
    // Get yesterday date
    var date = new Date(new Date().getTime() -
24*60*60*1000);
    // Convert to ISO time format
    date = date.toISOString().slice(0, 10);

    // Get year, month and day from String
    const year = date.toString().split("-")[0]
    const month =  date.toString().split("-")[1]
    const day = date.toString().split("-")[2]

    // Return in correct format
    return year + "-" + month + "-" + day;
}
```

Figure 71: *JavaScript function that returns yesterday's date*

Case 5: **Accessing the API**
All data that was received from Polygon.io has been converted in the form of an array, and after iterating through all received data, using the function fetchPopularStocks() necessary data has been extracted from this API. As seen in **Figure 71** only specific news has been extracted according to the requested information by the user. The front-end team could not implement continuous data fetching due to the limited API calls on the Polygon.io API. Unfortunately, unlimited data fetching is only available with an upgraded Polygon.io plan.

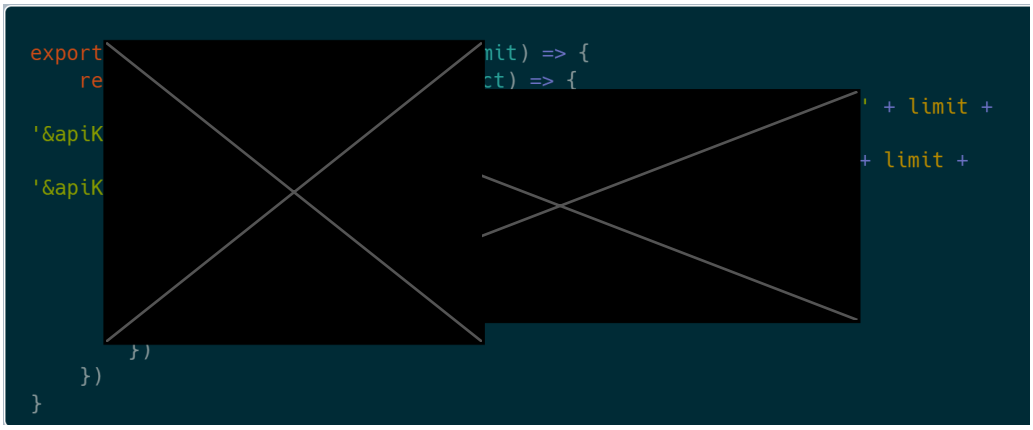


Figure 72: *JavaScript function that returns popular stocks*

Case 6: **CDN (Content Delivery Network)**
Allow for users to download local files based on the region that they are from. For example: An American user wants to download the web files, the network will decide to use the closest available CDN server to maximise speed performance.

Case 7: **Cloudfare**
Helps to prevent attacks onto the server by regulating users' requests onto the server when a user tries to connect, if too much activity is detected such as a bot attack (DDOS), it will refuse the connection to take place.

Case 8: **Keeping the application alive**
Due to the nature of SSH sessions and process management, the web server automatically ended its process after the user closed or left the SSH session. After research we found a program (Screen) that would allow us to keep the process running after closing the SSH session. (Azam 2021)

Case 9: **Crashes on requesting unavailable data**
During several tests on prototypes, we noticed that the API server crashes when signing into an account with an email that doesn't exist in the collection. The team decided to add additional checks in the request handler and return a valid JSON response with an error if this situation occurs. The checks have been made using the length of the return object from MongoDB.

Case 10: **Prevent users from accessing Index after signing in**
After we did some user testing on the public version of the web app, we got feedback that users were still be able to go back to the index page after accessing it. We prevented this by checking the authentication token of the user before rendering the Index page component and redirect the user using the Navigate package to the home page.

Case 11: **Unavailable Data**
Because of the conceptual and technical problems the back-end team faced during the development of the software, the front-end team found it difficult to implement certain proposed features. There was a delay in retrieving important information such as a response/JSON object from the back-end team. After increasing communication and talks between the two teams, we successfully decided on creating demo JSON files (**Figure 72**) to implement the missing features.

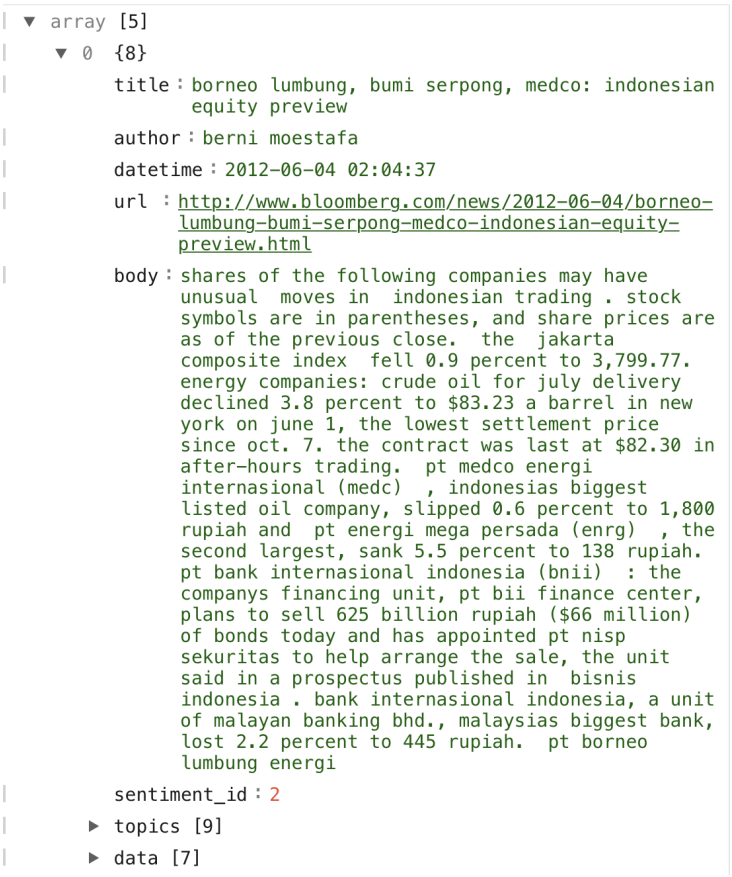


Figure 73: *An object of the searchResults Demo JSON*

Conclusion and summary

Back-end

In conclusion, the team has designed and implemented a pipeline that takes financial news data in the form of text: cleans it, performs topic clustering, sentiment analysis and outputs co-occurrent price trends for all available financial assets.

In the process the team has learned about natural language processing tasks including pre-processing, clustering of text data, financial sentiment analysis, BERTopic model, LDA and finBERT. In addition, the team has become familiar with analysing financial data in the form of tickers, retrieving upward and downward price trends. To aid us with the aforementioned work, the team has used Python supplemented by a number of libraries. These include: pandas, numpy, pymongo, dask, sentence-transformers, gensim, corpora, tqdm, pytorch, sklearn, bertopic, os, re, plotly, matplotlib, yfinance, datetime and nltk.

As a result, this team produced a functional demonstration of the proposed concept in the initial report (Biriukov et al. 2022). The main pitfall of the back-end implementation was the lack of up to date financial news data. This was a significant issue because the overall market is susceptible to a constant change, however the financial news the team used to build the co-occurrence algorithm was outdated. Any changes to the market since the end of the timeline of the data that was used (2013) would not be accounted for. This is an issue because the analysis of the outdated news will not be relevant to the current market. The team believes the pipeline will work sufficiently on a contemporary dataset.

Due to the time constraints and complexity of the project, the back-end team has not been able to effectively connect the pipeline to the front end web application. However, a sample output has been used to illustrate the functionality of the web application. The team intends to approach this problem in future work.

Front-end

The frontend team designed and implemented a prototype of the web application that uses modern technology to offer clients an organised web layout with most of the proposed features. During that process, the team learned how to set up a new React project, integrate different libraries such as TailwindCSS and HeadlessUI to design, animate and make user interfaces interactive. The team managed to develop the application in a structured way, so that changes to pages, components and functionality of features could be easily changed during the project's lifetime and in the future after making a public version available.

Besides working on the web application and implementing a custom API, the frontend team also configured a Linux Server that's hosted on an Azure Cloud. This includes setting up the correct permissions for users within the system, maintaining system updates and installing required software to run the different services.

Throughout the development, the team has researched best ways to document guidelines on the usage of the API Server with the frontend application and securely storing user sessions using technologies like Cookies, JWT tokens and API keys.

In the process of adding the Search and Dashboard page, the team has researched third-party API services to fetch financial data such as the current price of a stock. Since most API services require the symbol of a stock, rather than the name of the company, we proposed the idea of searching through a database (handled by the API service) of companies to retrieve the most accurate symbol based on the search query. This team was able to use the symbol to fetch the price of yesterday's close and additional information about the company after that and re-used some of the functions in other pages such as in the widgets on the Dashboard.

Additional features such as showing graphs of certain stocks couldn't be implemented in the given time frame, however the team managed to create low and high fidelity designs for the implementation of graphs and researched several libraries and guides on how to implement these with the data that was available to us.

Overall

Finally, the team has undertaken an ambitious project and effectively collaborated on numerous tasks to produce a demonstration of the intended product. For both the front and back end teams, a great deal of research, learning and implementation of sophisticated techniques has been carried out. Not only have all team members gained technical skill in specific disciplines but also extensive experience in teamwork, communication and time and resource management. Having laid the foundations for this software project in the planning report (Biriukov et al. 2022), the reality of implementing an ambitious piece of work has been made apparent through the numerous challenges and obstacles the team has had to overcome throughout the process.

References

1. n.d. PostCSS: Sass's New Play Date. https://www.toptal.com/front-end/postcss-sass-new-play-date.
2. Al-jabery, Khalid, and Donald Wunsch II. 2022. "Computational Learning Approaches to Data Analytics in
3. Biomedical Applications." sciencedirect. https://www.sciencedirect.com/topics/engineering/clustering-algorithm.
4. Araci, Dogu. 2019. "FinBERT: Financial Sentiment Analysis with Pre-trained Language Models." arxiv. https://arxiv.org/pdf/1908.10063.pdf.
5. Aroussi, Ran. n.d. "yfinance · PyPI." PyPI. Accessed April 7, 2022. https://pypi.org/project/yfinance/.
6. "Assigning categories to tweets using keyword matching." n.d. investigate.ai. Accessed April 7, 2022. https://investigate.ai/bloomberg-tweet-topics/assigning-categories-to-text-using-keyword-matching/.
7. "Assigning categories to tweets using keyword matching." n.d. investigate.ai. Accessed April 7, 2022. https://investigate.ai/bloomberg-tweet-topics/assigning-categories-to-text-using-keyword-matching/.
8. Azam, Shehroz. 2021. "How Do You Keep a Terminal Session Alive?" Linux Hint. https://linuxhint.com/keep-a-terminal-session-alive/.
9. "A beginner's guide to using Git when working with a team for the first time." 2020. DEV Community . https://dev.to/gladyspascual/a-beginner-s-guide-to-using-git-when-working-with-a-team-for-the-first-time-1hba.
10. Biriukov, D., B. Grey, J. Lange, C. Gauthier, A. Ziborovs, and A. Nasseur. 2022. "Computing Project 2 Report "MarketGeek."" Google Docs. https://docs.google.com/document/d/12o6GJVISnNJjBqxX8dpdIncuyENE-eANLWF7BO9wiQE/edit#heading=h.6c20tzh3rxzp.
11. Blei, David, and John Lafferty. n/a. "Correlated Topic Models." NeurIPS Proceedings. http://papers.neurips.cc/paper/2906-correlated-topic-models.pdf.
12. Blei, David, Andrew Ng, and Michael Jordan. 2003. "Latent Dirichlet Allocation." Journal of Machine Learning Research. https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf.

13. Brown, Justine. 2015. “Companies using sentiment-analysis software to understand employee concerns.” CIO Dive. https://www.ciodive.com/news/companies-using-sentiment-analysis-software-to-understand-employee-concerns/407357/.
14. Campello, Ricardo, Peer Kröger, Jörg Sander, Arthur Zimek, and University of Southern Denmark. 2020. “Density-based clustering.” findresearcher.sdu.dk. https://findresearcher.sdu.dk:8443/ws/files/171664269/widm.1343.pdf.
15. “Creating a React Project and integrating TailwindCSS — Part 1.” 2022. Medium. https://medium.com/@jjlange/creating-a-react-project-with-tailwindcss-and-headlessui-d67433b0f5b6.
16. “CUDA Toolkit.” n.d. NVIDIA Developer. Accessed April 5, 2022. https://developer.nvidia.com/cuda-toolkit.
17. Dieng, Adji, Francisco Ruiz, and David Blei. 2019. “Topic Modeling in Embedding Spaces.” arxiv. https://arxiv.org/pdf/1907.04907.pdf.
18. “Download the Node.js source code or a pre-built installer for your platform.” n.d. Download the Node.js source code or a pre-built installer for your platform. https://nodejs.org/en/download/.
19. Dua, Sejal. 2021. “NLP Preprocessing and Latent Dirichlet Allocation (LDA) Topic Modeling with Gensim.” Towards Data Science. https://towardsdatascience.com/nlp-preprocessing-and-latent-dirichlet-allocation-lda-topic-modeling-with-gensim-713d516c6c7d.
20. “Exponential Moving Average - Lag.” 2008. Elite Trader. https://www.elitetrader.com/et/threads/exponential-moving-average-lag.124782/.
21. “Facebook suffers $230bn wipeout in biggest one-day US stock plunge.” 2022. The Guardian. https://amp.theguardian.com/technology/2022/feb/03/facebook-stock-shares-meta-mark-zuckerberg.
22. Grootendorst, Maarten. 2022. “BERTopic: Neural topic modeling with a class-based TF-IDF procedure.” https://arxiv.org/. https://arxiv.org/pdf/2203.05794.pdf.
23. Grootendorst, Maarten. n.d. “Neural topic modeling with a class-based TF-IDF procedure.” arxic. https://arxiv.org/pdf/2203.05794.pdf.
24. Gupta, Shashank. n.d. “Sentiment Analysis: Concept, Analysis and Applications | by Shashank Gupta.”

Towards Data Science. Accessed March 31, 2022. https://towardsdatascience.com/sentiment-analysis-concept-analysis-and-applications-6c94d6f58c17.

25. “HDBSCAN, Fast Density Based Clustering, the How and the Why - John Healy.” 2019. YouTube. https://youtu.be/dGsxd67IFiU.
26. “https://polygon.io/docs/stocks/getting-started.” n.d. Stocks API Documentation.
27. “JSON Web Token Introduction - jwt.io.” n.d. JWT.io. Accessed April 7, 2022. https://jwt.io/introduction.
28. Li, Susan. 2018. “Topic Modeling and Latent Dirichlet Allocation (LDA) in Python.” Towards Data Science. https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24.
29. Malzer, Claudia, and Marcus Baum. 2021. “A Hybrid Approach To Hierarchical Density-based Cluster Selection.” arxiv. https://arxiv.org/pdf/1911.02282.pdf.
30. Mone, G. 2002. “colour Images More Memorable Than Black and White.” www.scientificamerican.com. https://www.scientificamerican.com/article/color-images-more-memorab/.
31. “News API.” n.d. News API. Accessed April 7, 2022. https://newsapi.org/docs.
32. Nielsen, Jakob. 1994. “Heuristic Evaluation: How-To: Article by Jakob Nielsen.” Nielsen Norman Group. https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/.
33. “n Install Tailwind CSS with Create React App.” n.d. n Install Tailwind CSS with Create React App. https://tailwindcss.com/docs/guides/create-react-app.
34. “npm-init Create a package.json file.” n.d. npm-init Create a package.json file. https://docs.npmjs.com/cli/v8/commands/npm-init.
35. “PostCSS: Sass’s New Play Date.” n.d. https://www.toptal.com/front-end/postcss-sass-new-play-date.
36. “THE PROS AND CONS OF TAILWIND CSS.” n.d. THE PROS AND CONS OF TAILWIND CSS. https://www.webdesignerdepot.com/2021/09/the-pros-and-cons-of-tailwind-css/.
37. “ProsusAI/finBERT: Financial Sentiment Analysis with BERT.” n.d. GitHub. Accessed March 31, 2022. https://github.com/ProsusAI/finBERT.
38. Remy, Philippe, and Xiao Ding. n.d. “philipperemy/financial-news-dataset: Reuters and Bloomberg.” GitHub. Accessed April 7, 2022. https://github.com/philipperemy/financial-news-dataset.
39. Rosner, F., A. Hinneburg, M. Roder, M. Nettling, and A. Both. 2012. “Evaluating topic coherence measures.”

Evaluating topic coherence measures. https://mimno.infosci.cornell.edu/nips2013ws/nips2013tm_submission_7.pdf.

40. Schmitt, Kirsten R. n.d. “What are the main advantages and disadvantages of using a Simple Moving Average (SMA)?” Investopedia. Accessed April 7, 2022. https://www.investopedia.com/ask/answers/013015/what-are-main-advantages-and-disadvantages-using-simple-moving-average-sma.asp.
41. Stanford edu. 2009. “Introduction to Information Retrieval.” Introduction to Information Retrieval. https://nlp.stanford.edu/IR-book/pdf/16flat.pdf.
42. “Stock news analysis.” n.d. Kaggle. https://www.kaggle.com/datasets/miguelaenlle/massive-stock-news-analysis-db-for-nlpbacktests.
43. “SyntheticEvent.” n.d. SyntheticEvent.

“Tickers.” n.d. Query all ticker symbols which are supported by Polygon.io. This API currently includes Stocks/Equities, Crypto, and Fore. https://polygon.io/docs/stocks/get_v2_snapshot_locale_us_markets_stocks__direction.

44. Vats, Rachit. n.d. “Getting Started | The Polygon.io Stocks API provides REST endpoints that let you query the latest market data from all US stock exchanges. You can also find data on company financials, stock market holidays, corporate actions, and more.” Polygon.io. Accessed April 7, 2022. https://polygon.io/docs/stocks/getting-started.
45. “What is .ENV files ?” n.d. What is .ENV files ? https://malware.expert/general/what-is-env-files/.
46. “What Is Jira?” n.d. What Is Jira? https://www.productplan.com/glossary/jira/#:~:text=Jira%20is%20a%20software%20application,%2C%20epics%2C%20and%20other%20tasks.
47. “What Is React?” n.d. https://reactjs.org/tutorial/tutorial.html#:~:text=React%20is%20a%20declarative%2C%20efficient,class%20ShoppingList%20extends%20React.
48. Whye, Yee, and Step Guide. 2021. “Part 2: Topic Modeling and Latent Dirichlet Allocation (LDA) using Gensim and Sklearn.” Analytics Vidhya.

https://www.analyticsvidhya.com/blog/2021/06/part-2-topic-modeling-and-latent-dirichlet-allocation-lda-using-gensim-and-sklearn/.

49. Zuma. 2013. “The psychology of colour - why are so many websites blue?” Zuma. https://www.zuma.co.nz/news/the-psychology-of-colour-why-are-so-many-websites-blue.

Appendix

Submission of the Report

- *PDF of report*
- *Software source code*
- *Version control logs (on GitHub)*

- [redacted]

- *GitHub:*
  [redacted]

Work history

- 21st Jan - Set up microsoft azure server for work environment
    - Set up user accounts for everyone
    - Set up Jupyter Notebook
    - Set up firewall on the cloud platform (azure), and on Ubuntu VM
    - Created home dirs and permissions for all users
- Setup Github repo's, frontend, back-end and api server, included readme, helped other team members learn Git and React
- Setup Jira for kanban, connected to github, has back reporting, apps to install on top
- API server, created the server file to host http server (the server itself), which is currently private
- API server, set up MongoDB on local machines, development server (azure)
- Following API features: create accounts, encryption for password (bcrypt), delete user accounts, authenticate using email and password (will be hashed), list users, environment file, small program to test all these features.
- Set up React.js project on local machines, front end web application started, currently the prototype of the index page is finished, a playful design in light and dark.
- Currently working on the identification
- Added private key authentication to the API Server to prevent unauthorised requests from others. The private key is set in the environment file (.env) in the root folder of the Web and API Server project.
- Added JWT (JSON Web Token) authentication to the API Server that creates tokens for incoming authentication requests from the web application. An access key is set in the environment file of the API Server that must be kept private as well.
- The API Server checks for the token in a separate new endpoint called "**/api/v1/user/checkToken**" and uses a middleware function called authToken to verify it. It uses its internal function and the pre-defined access key for that. A new access token is being created using the access key and the JWT "sign" method after a log-in request has been sent to the API Server and verified.
- The React application features a new Modal component that includes some basic configuration to change its title, description and name of the buttons. Using an onClick event, the user can open the Modal by pressing a button on the Index page. The content in the email and passwords fields are temporarily saved in a State and being proceeded using a library called Axios after the user submits the form. Axios sends a request to the API Server and handles the response accordingly.

- A new (/home) route allows the user to view a protected page within the React application. Users are required to be authenticated in order to view the page, otherwise they will be redirected back to the Index page where they will have to log in.
  The React application sends a request using Axios to the new API Server endpoint called "checkToken" that will check if the given token (saved in the local storage of the client's browser) is valid using JWT.

- Bug: API Server crashes if a client tries to log in using an email that doesn't belong to any account in the MongoDB collection.
- Implemented Dashboard functionality and Search Page with Artem
- Fixed bug for logged in users that are trying to access the index page
- UI tweaks to finialise product
- Updating GitHub with latest version

- Learning React
- Learning Github
- Course on Udemy for React
- Set up a development environment Node >= 14.0.0 and npm >= 5.6 on my machine in order to use the latest JavaScript features, provided by Node.js for developer experience, and optimise our app for production. Created components for User Testing page
- Set up MongoDB locally
- Installed GitHub, created access to the project files.
- Created page with low fidelity components for User testing.
- Created interactive components - buttons for low fidelity User testing.
- Working on overall web design with Anas and Justin.
- Implementing SearchPage and SearchBar with React.js
- Adding TailwindCSS to the SearchPage and SearchBar components
- Creating API rendering methods and functions.
- Updating GitHub with latest version

- Learning React
- Learning Github
- Course on Udemy for React
- Set up MongoDB locally
- Made small react views (components) to practice React before contributing to the main project on GitHub. Justin showed me how to style components using Tailwind and the JSX syntax.
- Worked closely with Justin and Artjoms to learn React further. Decided on purchasing an online React- course instead of using free resources on YouTube due to out-dated videos.
- Regular meetings with Justin to discuss first steps on GitHub and how to properly contribute in the following days/weeks. Explanation of how we manage pull-requests, Jira tickets and code reviews.

[redacted]

- Python scripts for sorting data
- Finding the right dataset for financial news, sourcing Reuters and Bloomberg
- Cleaning and sorting the financial news dataset
- Inserting the dataset into MongoDB
- Learning about ML and neural networks - [redacted]
- Using pandas, nltk, BERTopics, LDA in Python scripts for natural language processing, data management and analysis of the data [redacted]
- Create sentence_transformers from titles and inserted into the dataset.
- Using sentenceBERT model `'all-MiniLM-L6-v2'`
- sentenceBERT has given us the embeddings which represent the information from the titles as vectors.
- With the vectors we can compare the relevance of different article titles based on their location in vector space.
- Topic modelling with LDA initially on test data samples from the dataset.
- Topic modelling with BERTopics (reference and breakdown to come), creating clusters.
- Training and fine-tuning the BERTopic model and then predicted topics per article from the dataset.
- BERTopic uses BERT (bidirectional encoder representation from transformers, a deep learning model based on neural networks) and TF-IDF (topic frequency and inverse document frequency)
- BERT is a language representation model that is pre-trained and allows us to only apply one fine-tuning layer of training in order to get accurate results. This is a reason we believe it to be superior to LDA. The team will be comparing LDA to BERTopics as the best method for topic modelling.
- Migrating from azure to development server on Igor
- Back to azure, Igor was not compatible with Jupyter notebook
- Using local machine to do natural language processing as CUDA with RTX 3090 is much faster
- Found topic probabilities using LDA
- Used BERTopic to find topic clusters for generating the co-occurrences
- Lots of tweaking of the topic clusters to reduce total number of clusters for more chance of co-occurrences
- Reduced number of outliers by modifying certain parameters
- Found sentiment tensors using FinBERT
- HDBSCAN on the tensors found too many sentiment IDs, won't work for co-occurrences as too broad
- Converted sentiment tensors to IDs using the maximum value of each tensor, now we have only three possible sentiment IDs, better for co-occurrences
- Added topic IDs, topic words, sentiment ID to the overall news dataset

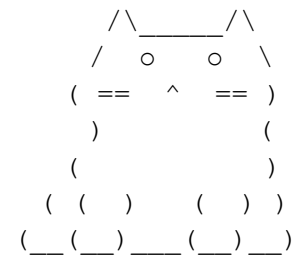
[redacted]

- Regex Python script for cleaning data and sorting
- Finding the right dataset for financial news, sourcing Reuters and Bloomberg
- Inserting yahoo financial tickers into the database
- Wrote a script to access starts and ends of upward and downward price trends by employing EMA crossover strategies
- Set up an environment on Igor
- Explored options beyond regular LDA
- Found BERTopic, compared it to LDA
- Worked with Benny to set up BERTopic and use it on our dataset
- Worked on tweaking the model to produce better results
- Worked on producing sentiment ids
- Wrote a script that finds price trends for a given ticker
- Wrote a script that finds co-occurrences, based on a query news article
- Found errors that resulted in inconsistent results in the output JSON, solved them
- Exported the JSON for the front end team consisting of 5 news articles with their respective statistics

[redacted]

- Python script grabbing daily news headlines (contemporary) related to business and exporting into csv
- Further work on the back-end, implementing the system of grabbing the daily news via news API set with a custom payload to grab relevant information, in this case, is financial from sources in the UK and USA. Data is then saved to a CSV and converted into a Mongo Database using pandas on the server so that it can be analysed by the sentiment analysis tool.
- Tweaked the implementation of the flow of grabbing news, so it is better integrated with other back-end members, allowing for a more organised csv output, along with a dataframe that better interacts with our existing mongo database.
- Manipulation of dataframe containing news JSOS and interaction with frontend infrastructure
- User testing, user testing and a bit more user testing

Thanks for reading :)



Gib good grade plz