UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

ISSN 0103-2569

MApp-IDEA: DESIGN AND ARCHITECTURE OF THE ANALYTICAL DATA EXPLORATION TOOL FROM APP REVIEWS

VITOR MESAQUE ALVES DE LIMA
JACSON RODRIGUES BARBOSA
RICARDO MARCONDES MARCACINI

Nº 443

RELATÓRIOS TÉCNICOS

São Carlos – SP
Jul./2023

MApp-IDEA: Design and Architecture of the Analytical Data Exploration Tool from App Reviews

Vitor Mesaque Alves de Lima[1], **Jacson Rodrigues Barbosa**[2], **Ricardo Marcondes Marcacini**[3]

[1]Faculty of Computing (FACOM)
Federal University of Mato Grosso do Sul (UFMS) – Brazil

[2]Institute of Informatics (INF)
Federal University of Goiás (UFG) – Brazil

[3]Institute of Mathematics and Computer Sciences (ICMC)
University of São Paulo – Brazil

vitor.lima@ufms.br, jacson_rodrigues@ufg.br, ricardo.marcacini@usp.br

Abstract. *Opinion mining for app reviews plays a crucial role in supporting software maintenance and evolution by analyzing user feedback. However, the manual analysis of a large amount of textual data poses a challenge. Machine learning-based methods have been employed to address this, but they primarily focus on historical user behavior and do not explore trend detection and risk classification of emerging issues. We describe the MApp-IDEA tool as an analytical data exploration tool that embodies the outcome of our research group's extensive research and development efforts over several years. Our approach presents a novel methodology to identify and prioritize emerging issues in app reviews. It involves the identification of defective software requirements and training predictive models to anticipate negatively evaluated requirements. Furthermore, we detect review issues, classify them using a risk matrix with prioritization levels, and monitor their evolution over time. By integrating these innovative components, the MApp-IDEA tool provides a comprehensive solution for app issue analysis and effective risk management. We explored our approach using over 6.6 million reviews across 20 domains, identifying and ranking nearly 270,000 issues. The results demonstrate the competitiveness of our approach and highlight the importance of extracting insights from user reviews to mitigate risks and improve app quality. The MApp-IDEA tool incorporates industry best practices, providing a powerful and user-friendly solution for app issue analysis and risk management.*

1. Introduction

Opinion mining for app reviews aims to analyze user comments on app stores to support software engineering activities, primarily software maintenance and evolution [Araujo et al. 2022]. One of the main challenges in maintaining software quality is promptly identifying emerging issues, such as bugs. However, manually analyzing these comments is challenging due to the extensive textual data.

Methods based on machine learning have been employed to automate opinion mining and address this issue [Araujo et al. 2020]. While recent methods have achieved promising results in extracting and categorizing issues from users' opinions, existing studies mainly focus on assisting software engineers in exploring users' historical behavior regarding app functionalities and do not explore mechanisms for trend detection and risk classification of emerging issues. Furthermore, these studies only cover part of the issue analysis process, from automated review collection to detection, prioritization, and analysis through an unsupervised approach [Lima et al. 2022].

We advance the state-of-the-art in opinion mining for app reviews by proposing an approach to identify and prioritize emerging issues. Our proposal introduces a two-fold approach that (i) identifies possible defective software requirements and trains predictive models for anticipating requirements with a higher probability of negative evaluation and (ii) detects issues in reviews, classifies them in a risk matrix with prioritization levels, and monitors their evolution over time.

We present the MApp-IDEA (Monitoring App for Issue Detection and Prioritization), an analytical data exploration tool that allows engineers to browse the risk matrix, time series, heat map, issue tree, alerts, and notifications. Our goal is to minimize the time between the occurrence of an issue and its correction, enabling the quick identification of problems. We processed over 6.6 million reviews across 20 domains to evaluate our proposal, identifying and ranking the risks associated with nearly 270,000 issues. The results demonstrate the competitiveness of our approach compared to existing models. We have proven that opinions extracted from user reviews provide crucial insights into app issues and risks and can be identified early to mitigate their impact. Our main contributions are briefly summarized below:

1. We introduce the prioritizing issues approach that automatically generates a risk matrix, combining sentiment analysis, clustering, and graph theory.
2. We present a method to generate the temporal dynamics of issues and the risk matrix using time series. Our method uses interval segmentation to calculate the frequency of problems in each time interval, where we are especially interested in intervals where abrupt changes occur.
3. Finally, we introduce an analytical data exploration tool that allows you to interactively browse the risk matrix, time series, heat map, and issue tree. Additionally, our analytic system triggers alerts and notifications.

This technical report is organized as follows. In Section 1, we provide an introduction to the concept of opinion mining for app reviews and its relevance to software engineering activities. Section 2 presents the MApp-IDEA method architecture, which is divided into five stages for detecting and prioritizing emerging app issues (EAI). In Section 3, we demonstrate the practical application of MApp-IDEA through an example. Section 4 offers a comprehensive technical and documentary overview of the design and architecture of the MApp-IDEA tool, focusing on key aspects such as component-based development, technologies schema, patterns, and the graphical interface. Finally, in Section 5, we provide concluding remarks summarizing the main findings and contributions of this research.

2. MApp-IDEA Method Architecture

To detect and prioritize EAI, we introduce MApp-IDEA. Our method is divided into five stages, as shown in Figure 1. First, we collect mobile app reviews from app stores via a web crawler. Second, these reviews are processed on a multilingual network model with over 100,000 nodes. We found the network node most associated with each review's snippets and searched the network for the best label to display. Third, we prioritize reviews in a risk matrix divided into three priority levels. Fourth, we model the risk matrix in time series to detect issue peaks over time and trigger alerts. Finally, we present the output of the previous stages in a user-friendly real-time interface through an interactive dashboard. An overview can be seen in Figure 1.

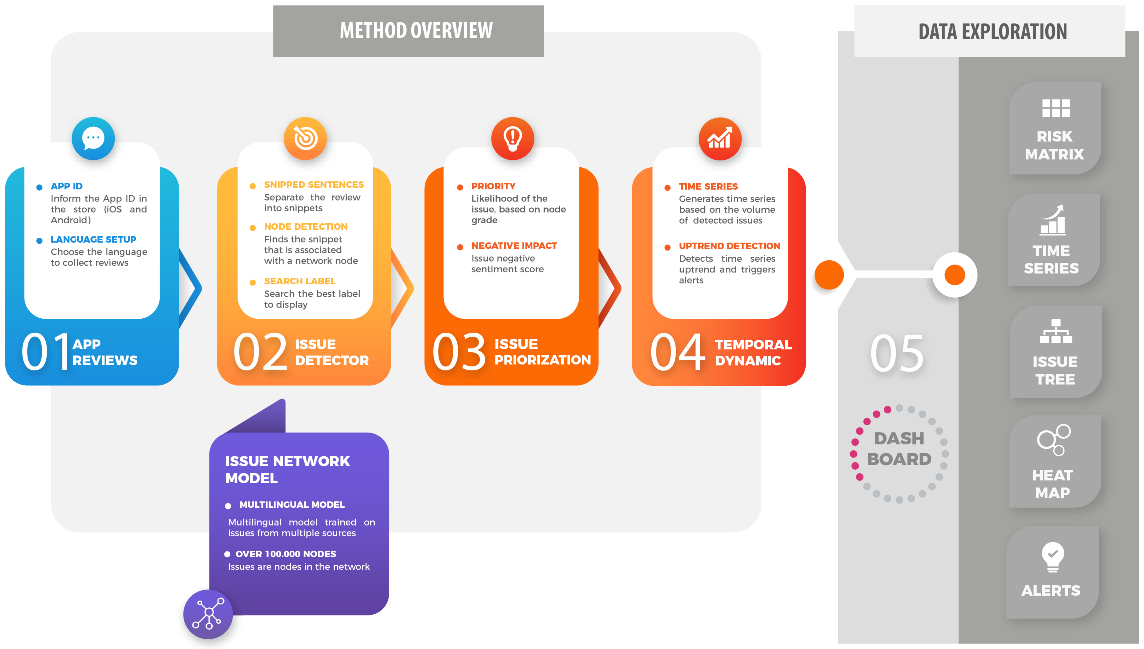

METHOD OVERVIEW
DATA EXPLORATION
APP ID
Inform the App ID in the store (iOS and Android)
LANGUAGE SETUP
Choose the language to collect reviews
01 APP REVIEWS
SNIPPED SENTENCES
Separate the review into snippets
NODE DETECTION
Finds the snippet that is associated with a network node
SEARCH LABEL
Search the best label to display
02 ISSUE DETECTOR
PRIORITY
Likelihood of the issue, based on node grade
NEGATIVE IMPACT
Issue negative sentiment score
03 ISSUE PRIORIZATION
TIME SERIES
Generates time series based on the volume of detected issues
UPTREND DETECTION
Detects time series uptrend and triggers alerts
04 TEMPORAL DYNAMIC
ISSUE NETWORK MODEL
MULTILINGUAL MODEL
Multilingual model trained on issues from multiple sources
OVER 100.000 NODES
Issues are nodes in the network
05
DASH BOARD
RISK MATRIX
TIME SERIES
ISSUE TREE
HEAT MAP
ALERTS

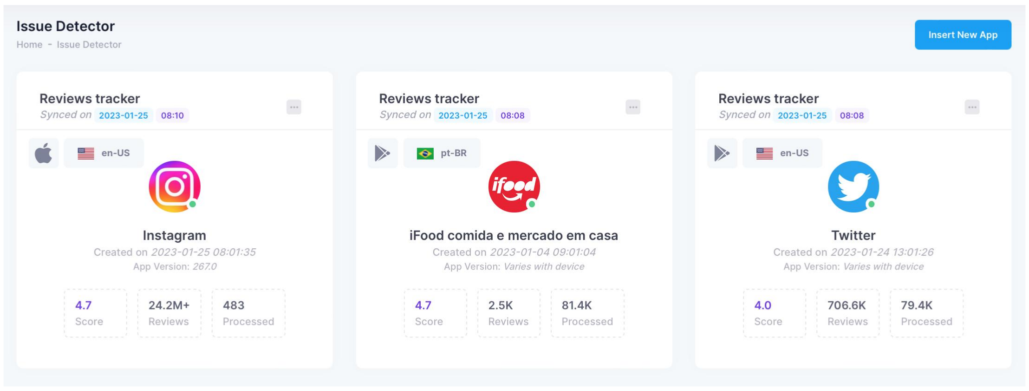
Figure 1. The architecture of the MApp-IDEA framework for detecting and prioritizing emerging issues

2.1. App Reviews

In the first stage of MApp-IDEA, raw reviews are collected from app stores using a web crawler tool via RESTful API. The app stores provide the textual content of the reviews, the publication date, and the rating stars of the reviews reported by users. In addition to reviews, we collect app metadata and metrics such as release dates, versions, number of installs, overall rating, and the total number of reviews. We use app metadata to drive overall performance and provide additional analysis of the temporal dynamics of metrics.

All configuration is done in this step through a user-friendly interface, as shown in Figure 3. The developer enters the app ID of the app store, the language, and the number of reviews to be processed. The crawler process runs in the background, and upon completion, the data is organized into the appropriate data structure for processing by the MApp-IDEA issue detection stage, as shown in Figure 2.

After app registration and initial processing, new app reviews are automatically processed and synchronized in the background over time, as shown in Figure 4. From this moment on, all app information is automatically tracked.


Issue Detector

Home - Issue Detector

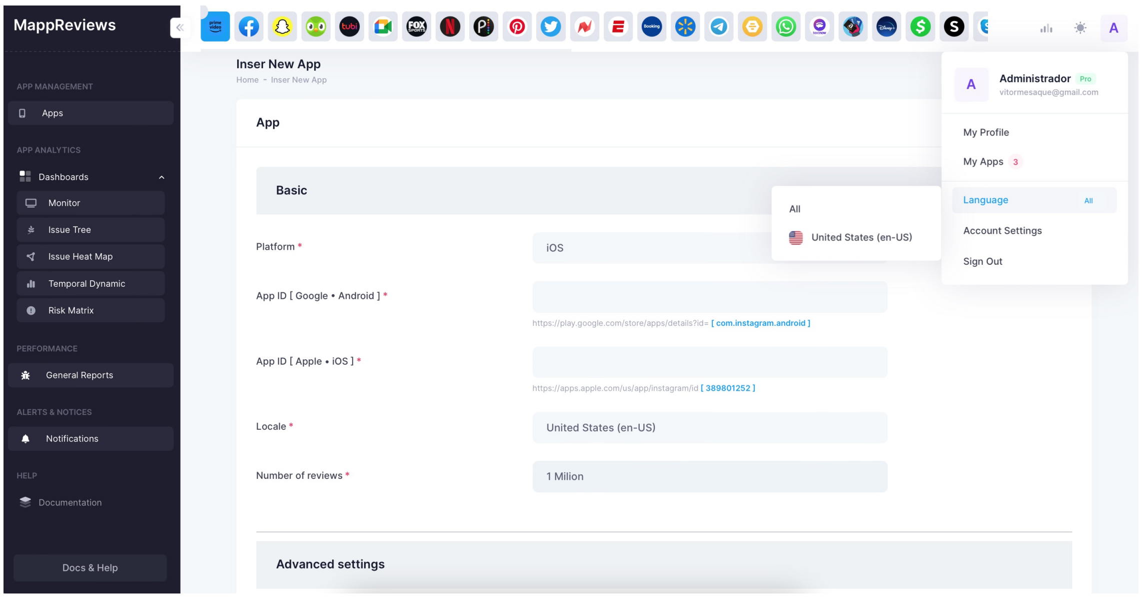
Insert New App

Reviews tracker
Synced on 2023-01-25 08:10
en-US
Instagram
Created on 2023-01-25 08:01:35
App Version: 267.0
4.7 Score | 24.2M+ Reviews | 483 Processed

Reviews tracker
Synced on 2023-01-25 08:08
pt-BR
iFood comida e mercado em casa
Created on 2023-01-04 09:01:04
App Version: Varies with device
4.7 Score | 2.5K Reviews | 81.4K Processed

Reviews tracker
Synced on 2023-01-25 08:08
en-US
Twitter
Created on 2023-01-24 13:01:26
App Version: Varies with device
4.0 Score | 706.6K Reviews | 79.4K Processed

Figure 2. Control panel to manage all apps



Inser New App

Home - Inser New App

App

Basic

Platform * iOS

All

United States (en-US)

App ID [Google • Android] *
https://play.google.com/store/apps/details?id= [com.instagram.android]

App ID [Apple • iOS] *
https://apps.apple.com/us/app/instagram/id [389801252]

Locale * United States (en-US)

Number of reviews * 1 Milion

Advanced settings

Figure 3. Initial app settings

2.2. Issue Detector

We explore word embeddings to build acyclic graphs for representing app issues. Word embeddings have recently been proposed to support NLP. They can calculate the semantic proximity between tokens and entire sentences, and the embeddings can be used as input to train the classifier. Additionally, graph-based methods have been widely used in several NLP tasks, such as text classification and summarization [Mihalcea and Radev 2011].

In our approach, we have an acyclic graph, where each issue is a vertex, and edges are defined by the similarity between issues based on the proximity of the issues vectors. In practice, we have a tree where each issue is a node, and similar issues with spelling variations or related issues are adjacent and connected by an edge. Therefore, we represent each issue through word embedding.

In graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path or, equivalently, a connected acyclic undirected graph. A forest is an undirected graph in which any two vertices are connected by at most one path, an acyclic undirected graph, or a disjoint union of trees [Williamson 2010].

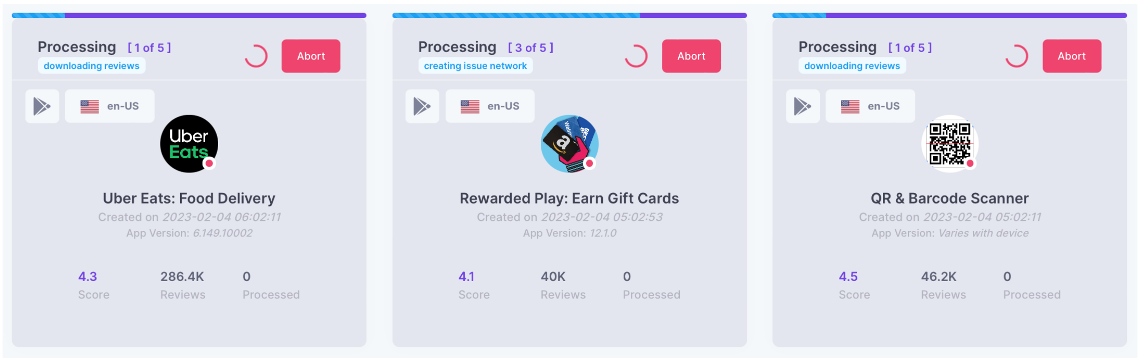


Figure 4. Synchronization process and automatic update of new reviews and issues

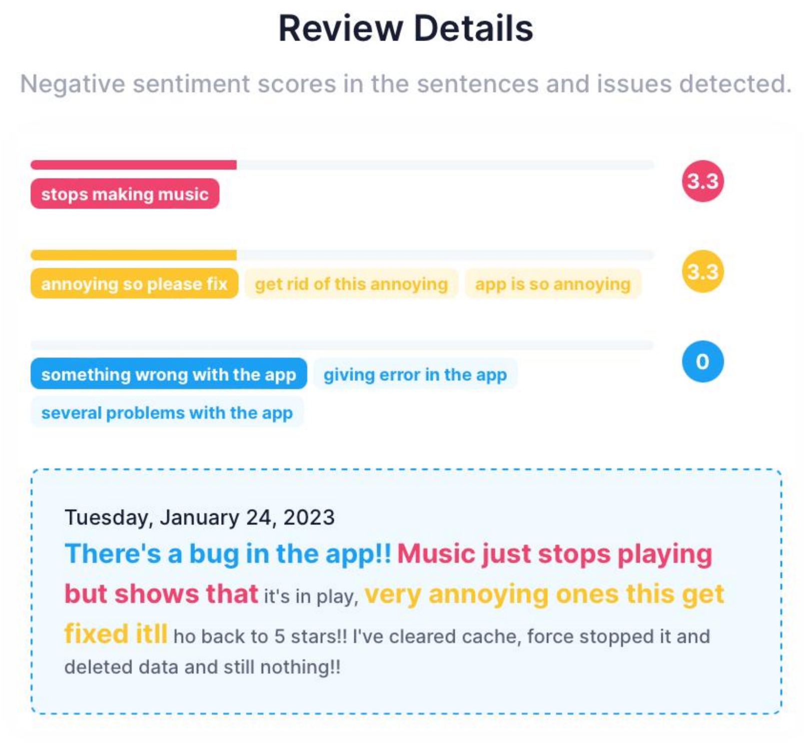
Our approach has a forest represented by the disconnected graph of the disjoint union of 3-tier trees. Each tree in the forest can have up to 3 tiers, wherein we have the best issue formations on the first and second tiers. The child nodes are connected to the parent nodes with better formations, but this is the same or related issue in practice. Therefore, we can group related issues and search the tree for the best node to display. We might have structurally weak issues in the tree, but we can search the tree and find the best good issue to display, as shown in Figure 5. However, a forest can have singleton graphs consisting of a single isolated node with no edges. In our approach, we can have issues that are not connected to any other.

To generate the issues graph, we used a multilingual model based on the BERT [Devlin et al. 2018], which allows the processing of reviews in several languages for issue detection. BERT is a contextual NLM where we can learn a word embedding representation for a token for a given sequence of tokens. For example, the vector space of embeddings preserves the proximity of similar issues but written in different ways by users such as "*problem with the payment*", "*i can't pay*", " *i can't complete the payment*" and "*payment error*".

In our architecture, we use a version of the BERT model called DistilBERT (a distilled version of BERT) [Sanh et al. 2019], which has the same architecture as the BERT model. In summary, the DistilBERT is a general-purpose pre-trained version of BERT, 40% smaller and 60% faster, that retains 97% of the language understanding capabilities [Sanh et al. 2019].

We use the Facebook AI Similarity Search (Faiss) [Johnson et al. 2017] to detect similarity between vectors. This library allows us to search high-dimensional vectors similar to each other using nearest-neighbor search implementations. Given a query vector, the similarity search returns a list of vectors closest to that vector in terms of Euclidean distance. Our issue detector classifier receives the correlation threshold and n-gram size parameters set to 0.8 and 7, respectively. The correlation threshold parameter defines the minimum similarity correlation between the vectors. A snippet is an issue if the distance between the most similar vectors exceeds the minimum correlation threshold. The n-gram size parameter is the maximum size of words that an issue can assume.

Summarly, we found the graph node most associated with each review's snippets and searched the 3-tier tree for the best label to display. From this, we can prioritize the



Review Details

Negative sentiment scores in the sentences and issues detected.

Tuesday, January 24, 2023

There's a bug in the app!! Music just stops playing but shows that it's in play, very annoying ones this get fixed itll ho back to 5 stars!! I've cleared cache, force stopped it and deleted data and still nothing!!

Figure 5. Details of issues detected in a review. For each review, MApp-IDEA extracts the review that is associated with the issue

most critical issues through sentiment analysis and particulars of the issues graph.

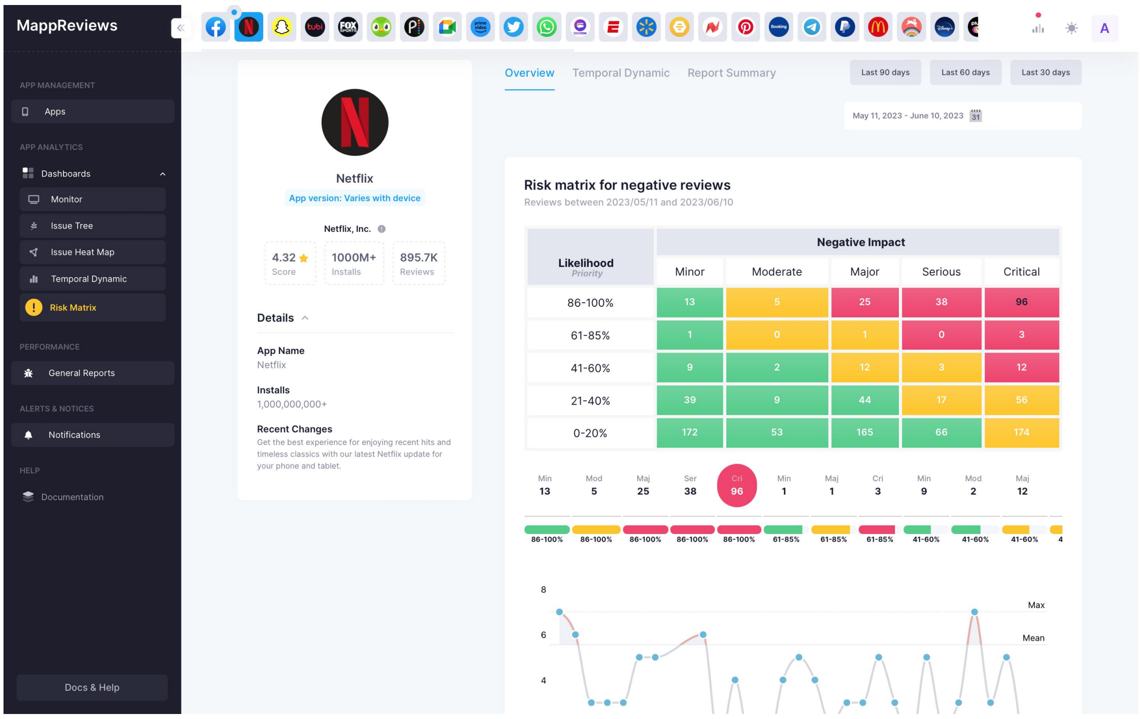
2.3. Issue Prioritization

After identifying issues, app developers must prioritize and address the most critical issues and ensure timely software maintenance and evolution. We propose prioritizing issues through an automatically generated risk matrix combining sentiment analysis, clustering, and graph techniques. Therefore, given an app and its reviews, we summarize reviews with one or more issues into a risk matrix, as shown in Figure 6.

We introduce the automatic generation of a risk matrix to predict issues with the most significant potential negative impact and probability of occurring. First, we assume that the likelihood of an issue i occurring is related to the similarity distance d_i of the issue i in relation to other nodes in the graph. We also assume that the issue's negative impact is related to how negative the issue's sentiment score is. Therefore, for each detected issue, we need a measure to calculate how many nodes are associated with the issue in the graph and the issue's sentiment score.

2.3.1. Negative Impact

To calculate the issue sentiment score, we use VADER (Valence Aware Dictionary for Sentiment Reasoning), a model used for text sentiment analysis sensitive to both polarities (positive/negative) [Hutto and Gilbert 2014]. VADER sentiment analysis relies on a dictionary that maps lexical features to emotion intensities known as sentiment scores.



Netflix

App version: Varies with device

Netflix, Inc.

4.32	1000M+	895.7K
Score	Installs	Reviews

App Name
Netflix

Installs
1,000,000,000+

Recent Changes
Get the best experience for enjoying recent hits and timeless classics with our latest Netflix update for your phone and tablet.

May 11, 2023 - June 10, 2023

Risk matrix for negative reviews

Reviews between 2023/05/11 and 2023/06/10

Likelihood *Priority*	Negative Impact				
	Minor	Moderate	Major	Serious	Critical
86-100%	13	5	25	38	96
61-85%	1	0	1	0	3
41-60%	9	2	12	3	12
21-40%	39	9	44	17	56
0-20%	172	53	165	66	174

Min	Mod	Maj	Ser	Cri	Min	Maj	Cri	Min	Mod	Maj	
13	5	25	38	96	1	1	3	9	2	12	
86-100%	86-100%	86-100%	86-100%	86-100%	61-85%	61-85%	61-85%	41-60%	41-60%	41-60%	4

Figure 6. Navigable risk matrix. By clicking on the cell, it is possible to browse the time series of each cell

The sentiment score of a text can be obtained by summing the intensity of each word in the text.

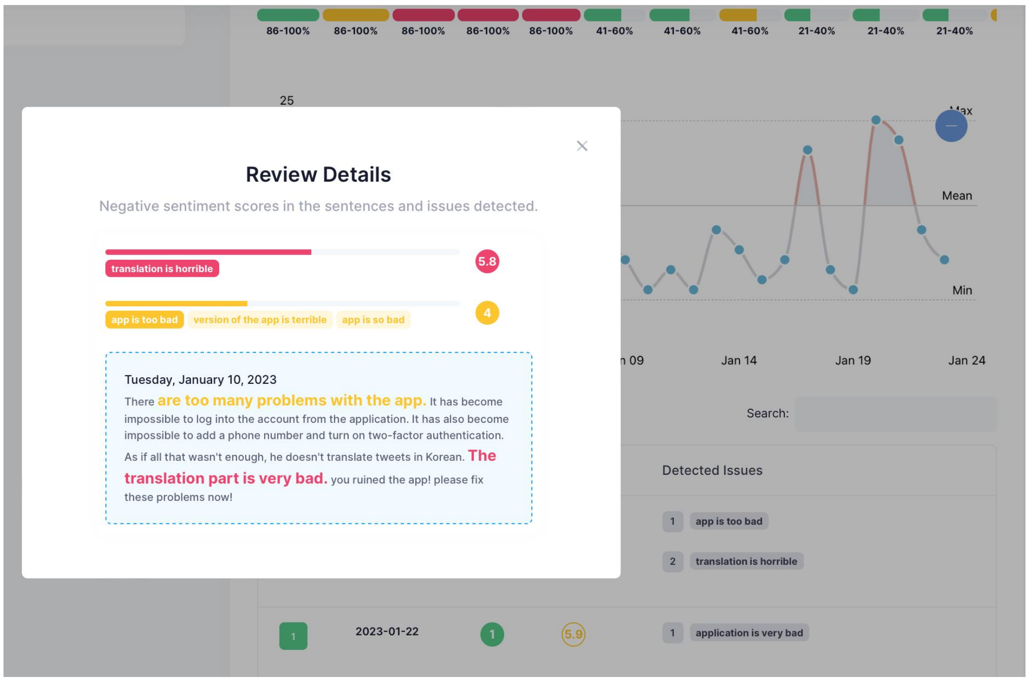
The compound score is a metric that calculates the sum of all the lexicon ratings which have been normalized between -1 (most extreme negative) and $+1$ (most extreme positive): (i) positive sentiment $(compoundScore \geq 0.05)$, (ii) neutral sentiment $(compoundScore > -0.05)$, and (iii) $(compoundScore < 0.05)$ negative sentiment $(compound_{score} \leq -0.05)$. For our risk matrix, we consider negative sentiment, i.e., the compound score less than or equal to zero $(C_s \leq 0)$. On the x-axis of the matrix is the discretization of the negative sentiment score into four levels.

To show the result more understandably, we present the negative impact in absolute (positive) values, ranging from 0 to 10, as shown in Figure 7.

2.3.2. Likelihood

On the y-axis (likelihood), we have the discretization of the degree of similarity of the issue with other issues in the graph in four levels. The higher the level, we have more related issues. The lower the level, the less similarity the issue has to other issues.

We use a partitional clustering strategy to calculate the issue degree measure in the graph. We cannot simply compute the degree of the node in the graph to generate the x-axis value because, in our model, we may have too many singleton graphs, which would unbalance the matrix. Therefore, we need to consider this to calculate the likelihood. Let V be the number of vertices in the graph. We group all vertices of the graph into $\sqrt{|V|}$



Review Details

Negative sentiment scores in the sentences and issues detected.

translation is horrible 5.8

app is too bad | version of the app is terrible | app is so bad 4

Tuesday, January 10, 2023

There **are too many problems with the app.** It has become impossible to log into the account from the application. It has also become impossible to add a phone number and turn on two-factor authentication. As if all that wasn't enough, he doesn't translate tweets in Korean. **The translation part is very bad.** you ruined the app! please fix these problems now!

Figure 7. Navigable risk matrix. By clicking on the cell, it is possible to browse the time series of each cell

clusters and define the x-axis value of each node as the size of its cluster, i.e., the number total number of issues that are in the cluster.

In partitional clustering, the main goal is to partition a set of examples into k groups, where the user typically determines the value of k. MApp-Reviews utilizes the K-means algorithm [MacQueen et al. 1967], which is a widely recognized and extensively employed method for partitional clustering, particularly in the analysis of textual data.

Formally, let $R = \{i_1, i_2, ..., i_n\}$ a set of issues, where each issue i is a m-dimensional real vector from an word embedding space. The K-means clustering aims to partition the n issues into k $(2 \leq k \leq n)$ clusters $C = \{C_1, C_2, ..., C_k\}$, thereby minimizing the within-cluster sum of squares as defined in Equation 1, where μ_i is the mean vector of all issues in C_i.

$$\sum_{C_i \in C} \sum_{r \in C_i} \|r - \mu_i\|^2 \tag{1}$$

We defined the cluster number k as $\sqrt{n}$, where n is the total issues (nodes). Then, for each issue, we define its likelihood as the size of the cluster in which it is inserted. We then normalize this value between 0 and 1 using Min-Max (2). Therefore, the likelihood of the issue occurring is associated with the similarity between the nodes in the network, as a very frequent issue will have many occurrences, variations of writing, and different ways of describing the same problem by users.

2.3.3. Risk Matrix Construction

Our algorithm for calculating the risk matrix places the reviews with one or more issues in their respective cell [i,j] of the matrix according to the thresholds of negative impact and likelihood.

With the algorithm's output, we prioritized the analysis, observing the most likely and critical problems, as shown in Figure 6. The algorithm subdivides the matrix into three priority levels: level 1 (low), level 2 (medium), and level 3 (high).

2.4. Temporal Dynamic

Considering that the issue time series is a set of observations obtained sequentially over time, as illustrated in Figure 8, we can model the occurrence of issues over time in a time series to detect upward trends in their frequency, as shown in Figure 9. Formally, an issue time series I of size s can be represented as an ordered sequence of observations denoted as $I = (i_1, i_2, ..., i_s)$, where each observation i_t at time t belongs to the set of real numbers ($\mathbb{R}$) and represents the number of issues observed at that particular time point [Chatfield 2003].

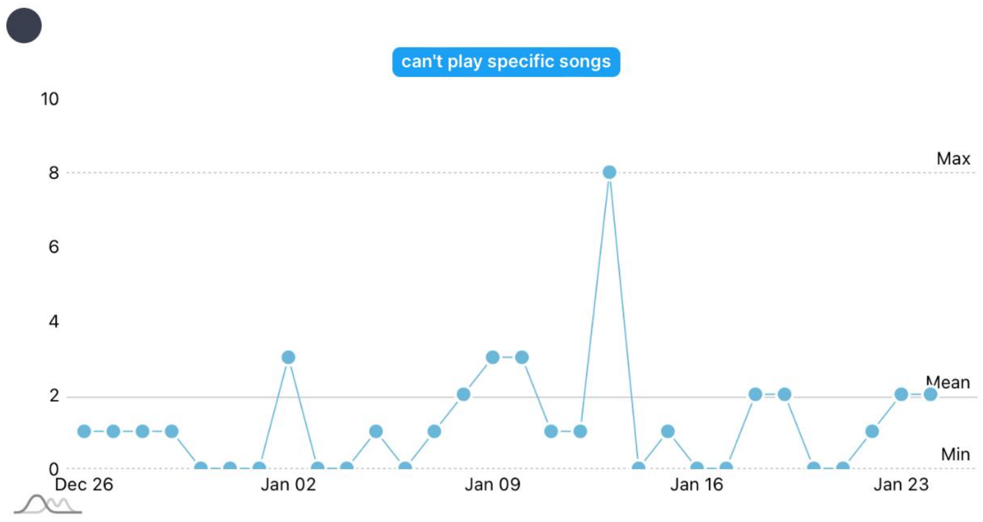


Figure 8. Issue time series "can't play specific songs"

In addition to capturing the temporal dynamics of issues, we also explore the temporal dynamics of the risk matrix. Once we have constructed the risk matrix, we examine its temporal behavior for each cell and prioritize level. Figure 10 visually depicts the time series of priority levels associated with the identified issues. The green, yellow, and red lines in the figure correspond to low, medium, and high priority levels, respectively. This analysis provides insights into how the risk levels evolve over time and allows us to monitor the changing prioritization of issues.

3. MApp-IDEA in Action

We present an example of MApp-IDEA's usage in this section.

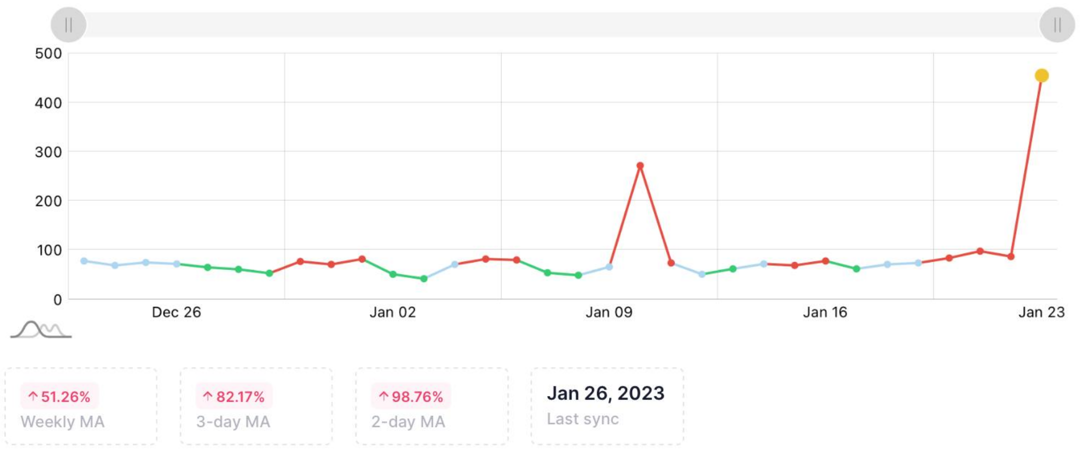


Figure 9. Time series of Spotify app issues with peak issues on January 10th and 23rd. Red lines indicate an uptrend in the time series.

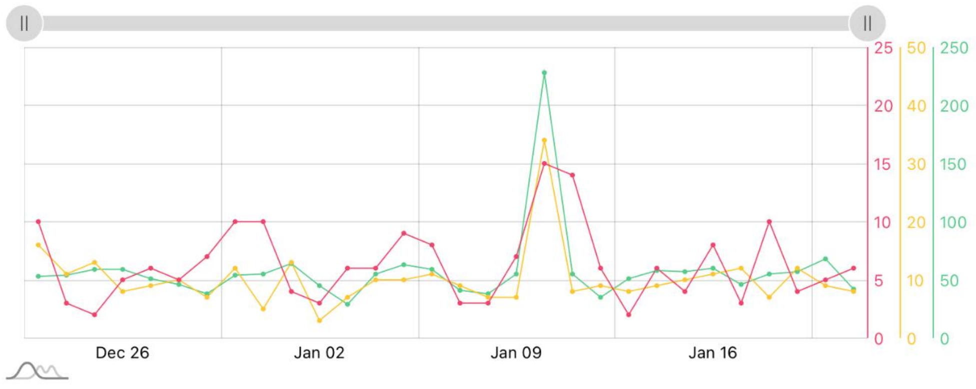


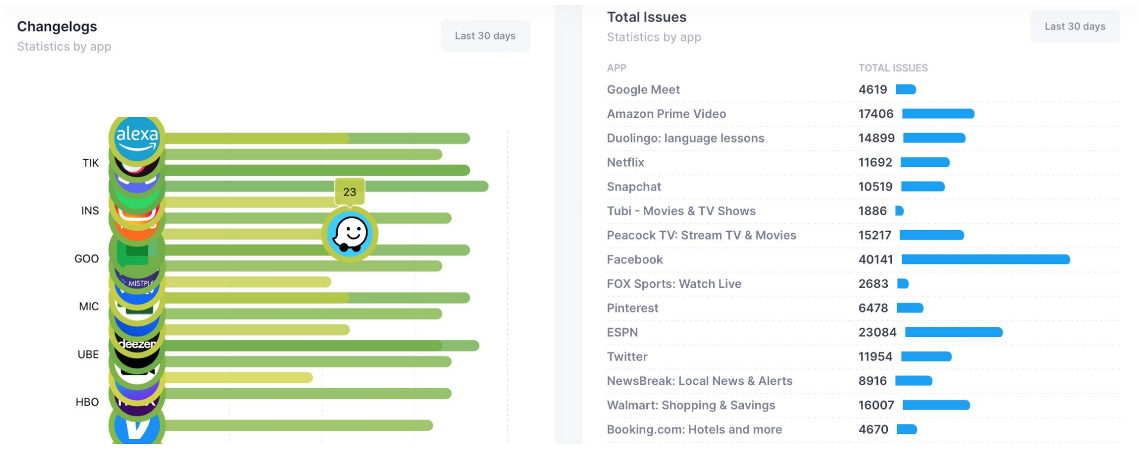
Figure 10. Time series of issues with three lines representing issues in priority levels. The colors green, yellow, and red represent levels 1, 2, and 3

3.1. Monitor

On the home screen, MApp-IDEA presents all monitored applications, with daily performance statistics of apps concerning comment volume and star rating, as shown in Figure 12. We also provide a summary of notifications and flips, the daily trend of top issues, the volume of issues detected for all apps, and changelog statistics, as shown in Figure 11 and Figure 13.

3.2. 3-tier Navigable Tree

In this visualization approach, interactive navigation across the three layers of the tree is facilitated, enhancing the understanding of the underlying data structure. The size (diameter) of each node in the visualization corresponds to the volume of issues associated with the subtree rooted at that node, as illustrated in Figure 15. Larger nodes indicate a higher frequency of issues. Furthermore, each distinct tree in the visualization is assigned a color, representing a group of related issues categorized within up to three levels, as illustrated in Figure 14. This visualization technique enables users to explore and analyze similar issue groups effectively.



Total Issues

Statistics by app

Last 30 days

APP	TOTAL ISSUES
Google Meet	4619
Amazon Prime Video	17406
Duolingo: language lessons	14899
Netflix	11692
Snapchat	10519
Tubi - Movies & TV Shows	1886
Peacock TV: Stream TV & Movies	15217
Facebook	40141
FOX Sports: Watch Live	2683
Pinterest	6478
ESPN	23084
Twitter	11954
NewsBreak: Local News & Alerts	8916
Walmart: Shopping & Savings	16007
Booking.com: Hotels and more	4670

Figure 11. Dashboard with daily performance information for each app on the star rating and number of reviews

3.3. Network Exploratory Heatmap

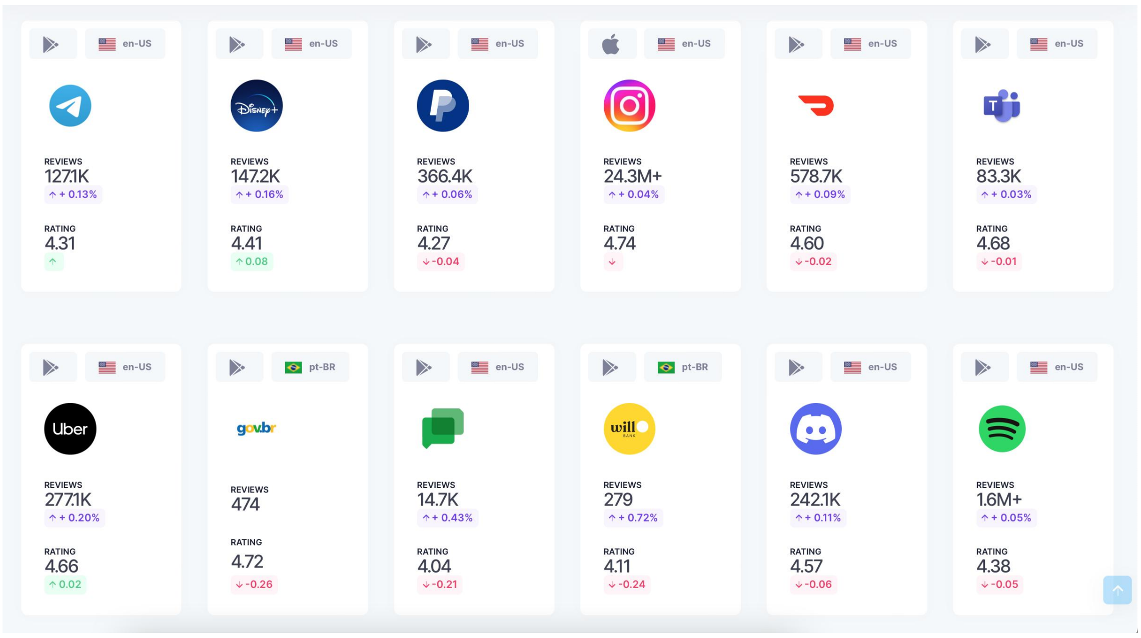
We developed an exploratory issue network called a heat map, which emphasizes issues with higher priority according to the volume of occurrences and degree of the node in the graph. In this form of visualization, we use the most representative nodes of the tree, from layers 1 and 2, to generate a new graph of issues with approximately 600 nodes.

Mathematically, a network is defined as an undirected graph $G = (V, E)$, formed by a set $V = (v_1, v_2, ..., v_n)$ of nodes (words). A set $E = (e_1, e_2, ..., e_n)$ of edges that are represented by an adjacency matrix A, whose elements A_{ij} are equal to 1 whenever there are an edge connecting nodes (words) i and j, and equal to 0 otherwise.

This new graph allows us to identify critical regions of the graph through the volume of issues related to each node and categorize the visualization into different critical levels. Each node of this graph is connected to similar nodes, calculated by the distance of similarity between the vectors. The “payment problem” node will be close to the “credit card problem” node, with an edge connecting these two nodes. We call this new graph a Heat Map, representing the critical nodes in 4 levels, from the least critical to the most critical, as shown in Figure 16 and Figure 17.

For example, consider app A, with 10,000 reviews in the last 30 days. After detecting the issues, we already know which tree nodes are related to the issues. The identified application A issues can belong to any of the three layers of the original tree, with layers 1 and 2 being the most representative. Suppose 200 issues of application A were detected. These detected issues are distributed in the graph according to their representation. Consider that 50% of these issues are related to the “app stopped working” node, 25% are related to the “payment problem” node, 15% are related to the “I can’t access my account” node, and 10% are related to other nodes. The Heat Map Graph has 600 nodes in all. However, these three detected nodes will be highlighted, as exemplified in Figure 18. In this example, the “app stopped working” will be marked with red color, flashing on the screen, and with a larger diameter than all the others.

The “payment problem” node will be marked red, but without flashing and with a smaller diameter. The “I can’t access my account” node will be marked in yellow. The


en-US
REVIEWS
127.1K
+ 0.13%
RATING
4.31

en-US
REVIEWS
147.2K
+ 0.16%
RATING
4.41
0.08

en-US
REVIEWS
366.4K
+ 0.06%
RATING
4.27
-0.04

en-US
REVIEWS
24.3M+
+ 0.04%
RATING
4.74

en-US
REVIEWS
578.7K
+ 0.09%
RATING
4.60
-0.02

en-US
REVIEWS
83.3K
+ 0.03%
RATING
4.68
-0.01

en-US
REVIEWS
277.1K
+ 0.20%
RATING
4.66
0.02

pt-BR
REVIEWS
474
RATING
4.72
-0.26

en-US
REVIEWS
14.7K
+ 0.43%
RATING
4.04
-0.21

pt-BR
REVIEWS
279
+ 0.72%
RATING
4.11
-0.24

en-US
REVIEWS
242.1K
+ 0.11%
RATING
4.57
-0.06

en-US
REVIEWS
1.6M+
+ 0.05%
RATING
4.38
-0.05

Figure 12. Dashboard with daily performance information for each app on the star rating and number of reviews

other nodes in the network will be marked in blue or gray. Nodes marked in blue are nodes not detected by the issue detector for application A, with a high vertex degree, i.e., they are more strongly connected to other nodes. If these blue nodes turn yellow or red, it could be a severe problem. The nodes marked in gray were not detected by the issue detector and had a low vertex degree value, i.e., they are problems of lesser impact.

To calculate the threshold between the critical levels of the graph, we calculate the frequency of occurrence of each node. Then we normalize the frequency to obtain values between 0 and 1, using the Min-max scale, according to equation 2.

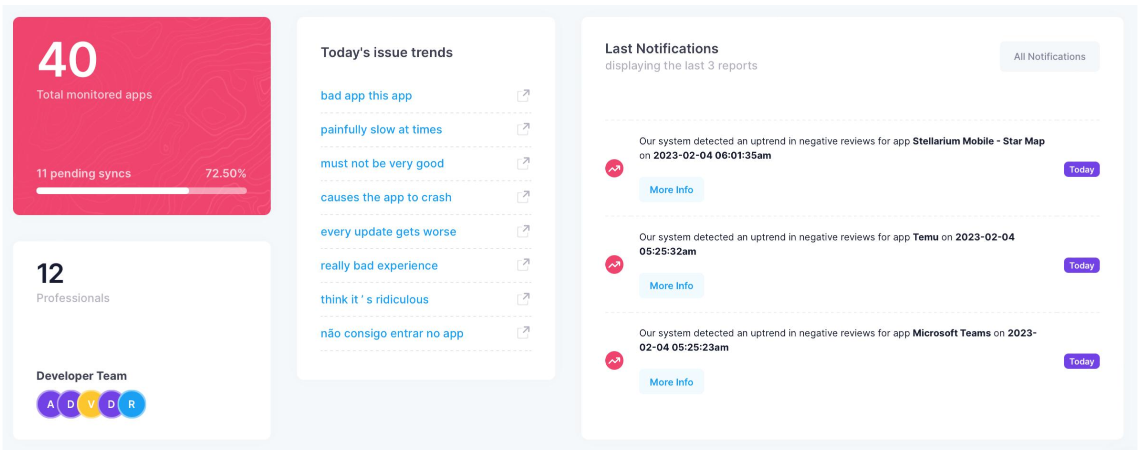
$$m = \frac{(x - x_{min})}{(x_{max} - x_{min})} \tag{2}$$

where m is the new value, x is the original value of the cell, x_{min} is the minimum frequency value, and x_{max} is the maximum value of the frequency.

Subsequently, the computed value of m is compared to a threshold value determined by the software engineer. The choice of the threshold influences the sensitivity of the heat map graph coloring. A lower threshold increases the level of sensitivity, resulting in more evident variations in the color scheme of the heat map graph, as shown in Figure 19.

The expectation is that denser regions at higher critical levels will receive prioritization. If there are numerous red dots clustered in a specific region, it indicates that app users are reporting a common and frequent problem. Such frequent issues tend to occur repeatedly, and spelling variations will be a "hub" in the network.

Our platform offers interactive navigation within the graph, allowing visualization of reviews associated with each node and access the top-n issues linked to those reviews.


40
Total monitored apps
11 pending syncs 72.50%

12
Professionals

Developer Team
A D V D R

Today's issue trends
- bad app this app
- painfully slow at times
- must not be very good
- causes the app to crash
- every update gets worse
- really bad experience
- think it ' s ridiculous
- não consigo entrar no app

Last Notifications
displaying the last 3 reports
All Notifications

Our system detected an uptrend in negative reviews for app **Stellarium Mobile - Star Map** on **2023-02-04 06:01:35am** Today
More Info

Our system detected an uptrend in negative reviews for app **Temu** on **2023-02-04 05:25:32am** Today
More Info

Our system detected an uptrend in negative reviews for app **Microsoft Teams** on **2023-02-04 05:25:23am** Today
More Info

Figure 13. Summary dashboard of top-n issues of the day for monitored apps

3.4. Interactive Risk Matrix

The risk matrix generated by our algorithm is presented in a dynamic and navigable way. Each cell of the matrix is clickable, where we can view the issues related to each cell and the reviews and sentences linked to the issues. We can also sort issues by priority, likelihood, and negative impact levels. For each cell in the matrix, we also display the temporal modeling, i.e., the evolution of that cell over time, as shown in Figure 20.

Additionally, we provide summary reports of issue priority levels, as shown in the screen example (Figure 21).

3.5. Time series

The MApp-IDEA presents different types of time series to observe the time evolution of issues and identify trends.

The trend detector time series is based on the moving average displayed in three colors on the lines. Red represents the uptrend, green represents the downtrend, and light blue represents stability, as shown in Figure 22.

We use a peak detection algorithm based on the dispersion principle (Z-score). If a new data point is a given x number of standard deviations from some moving average, the algorithm triggers $+1$ or -1. The system triggers an Uptrend alert if a $+1$ signal is returned. If a -1 signal is triggered, we alert a Downtrend.

Formally, a raw score x is converted into a standard score (Z-score) according to equation 3:

$$z = \frac{x - \mu}{\sigma} \tag{3}$$

where μ is the issues mean and σ is the issues standard deviation.

The absolute value of z represents the distance between this raw score x and the issues mean in standard deviation units. z is negative when the raw score is below the mean and positive when it is above.

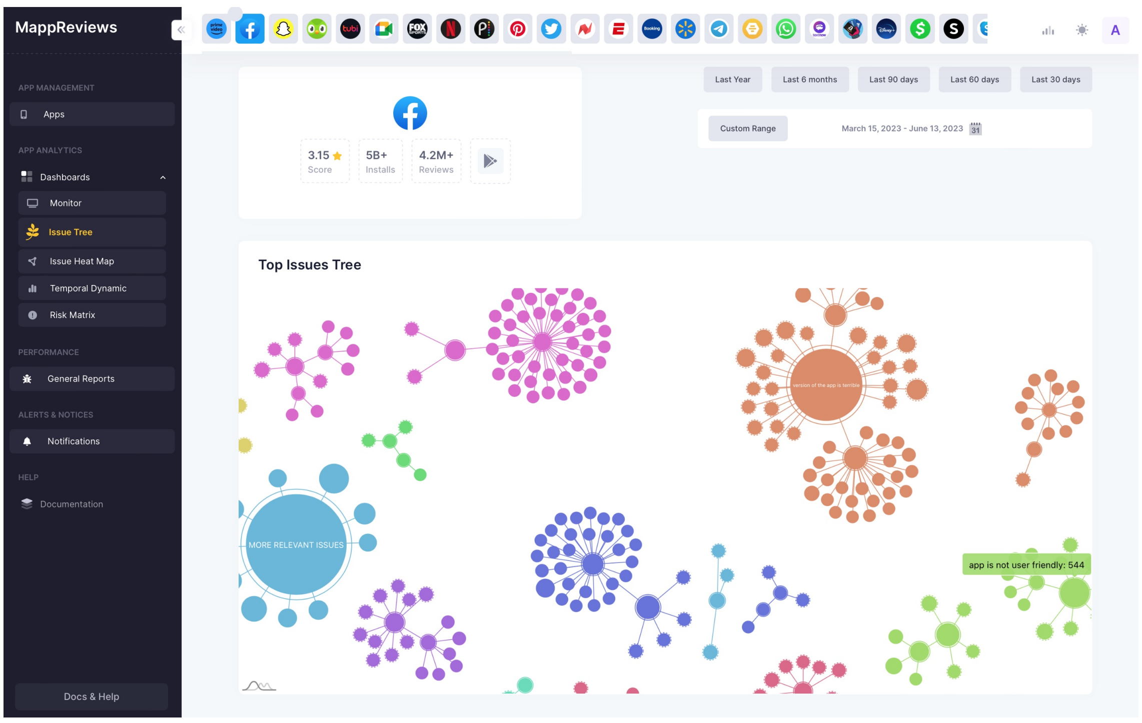


Figure 14. Facebook app's issue tree into a dynamic and interactive 3-tier structure between March 15 and June 13, 2023

The issue detector algorithm receives three parameters: (i) the moving window, (ii) threshold (the Z-score on which the algorithm signals), and (iii) influence (value between 0 and 1 that implies the influence of new signals on the average and in standard deviation).

With the risk matrix output, we generate a stacked area time series with three priority levels of issues stacked on each other. Therefore, we can compare the evolution of the whole and the contributions of individual parts over time, as shown in Figure 23.

In addition, we present trend detection signals, i.e., a time series emphasizing trend detection points. Red areas represent uptrends, and downtrends are represented by green areas, as shown in Figure 24. Each point in the time series has a complete report of all reviews and detected issues, as shown in Figure 25.

3.6. Notification System

Issue alerts occur when MApp-IDEA monitoring detects risks associated with an increased volume of issues. MApp-IDEA generates intelligent alerts to notify the team about changes, high-risk issues, or environmental failures, as shown in Figure 26.

Due to the way the issue detector module is implemented, even if there is no defective functionality related to a software requirement in the app, the MApp-IDEA alert system will still be able to identify failures in the environment, e.g., offline resources or processing web requests taking longer than usual, decreasing database latency, or app policy changes. These environmental issues are also frequently reported by users, although described differently.

The purpose of alerts is to quickly identify and resolve issues that affect app avail-

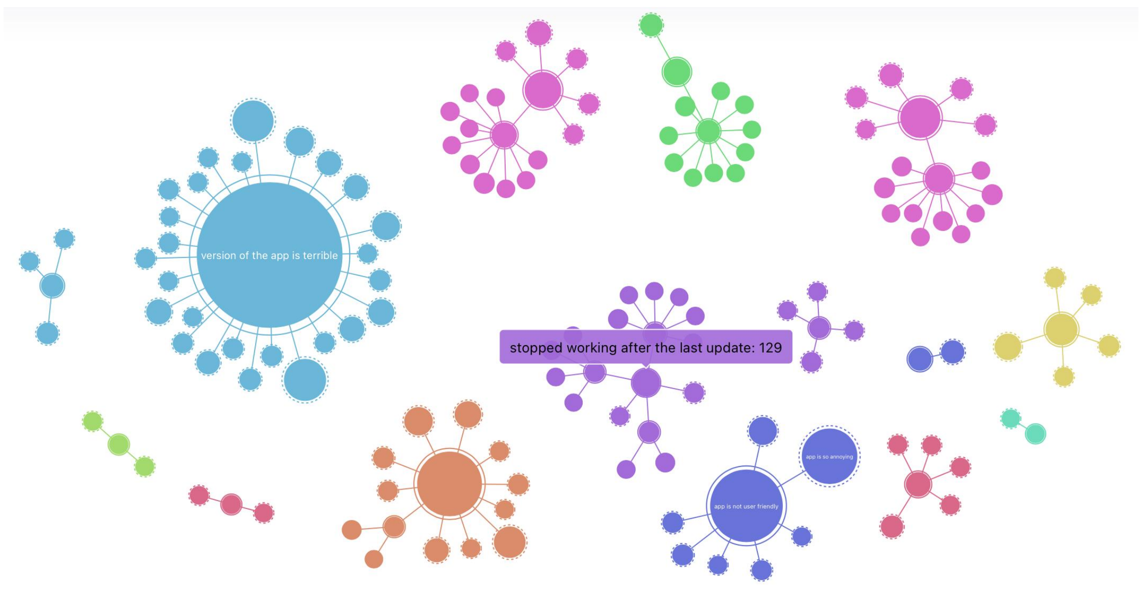

version of the app is terrible
stopped working after the last update: 129
app is so annoying
app is not user friendly

Figure 15. 3-tier tree dashboard

ability, speed, and functionality without manual monitoring. Using automatic features to monitor apps and generate alerts, developers can minimize downtime and reduce diagnostic time and the associated high cost.

3.7. Online Performance Reports

The MApp-IDEA dashboard generates a complete real-time report of the relationship between releases and issue peaks, reports on average updates, average issues detected per day, and the average interval between releases. In addition, it presents a summary of the total reviews collected and processed and the total number of issues detected. A sample screen is shown in Figure 27.

4. Design and Architecture of the Analytical Data Exploration Tool

Now, we provide a comprehensive technical and documentary overview of the architecture and implementation of our analytical tool designed for data exploration, called the MApp-IDEA tool.

The rest of this section is structured as follows. In section 4.1, we will discuss various system design and implementation aspects. We will start by presenting the technologies schema 4.2, which provides an overview of the key technologies components separated in a stacked layer schema. Next, we will delve into patterns 4.3, specifically architectural patterns and design patterns. We will explore the characteristics of these patterns and how they are used in the project. Section 4.4 explores the system's graphical interface, discussing user interface design principles and frameworks employed. Additionally, we will explore the RESTful Bus (4.5) architecture for communication, explaining its concept, advantages, and implementation details in the system.

4.1. Component-based Development

Software development is a complex process that can benefit significantly from adopting a component-based development approach. Component-based development focuses on

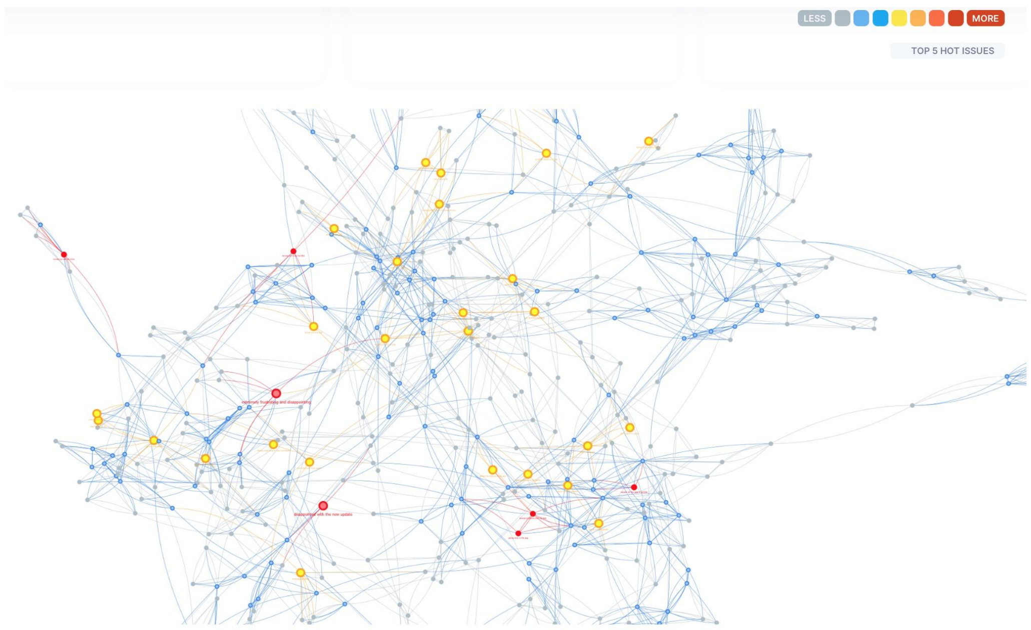
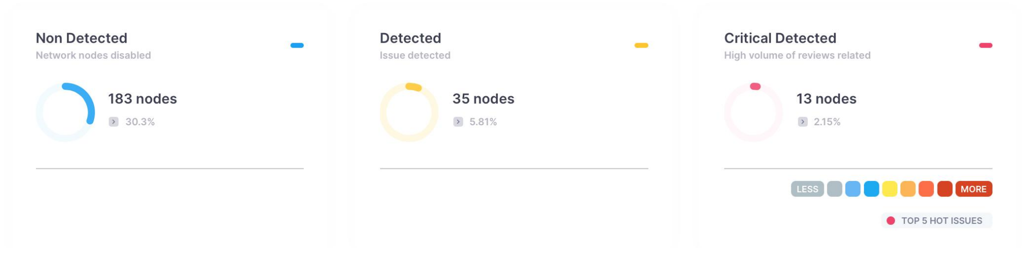


Figure 16. Heatmap network where each node represents an issue and is connected by similarity


Non Detected
Network nodes disabled
183 nodes
30.3%

Detected
Issue detected
35 nodes
5.81%

Critical Detected
High volume of reviews related
13 nodes
2.15%

Figure 17. Description of color variation in heatmap

building software systems by assembling reusable and self-contained components, each responsible for a specific functionality or service [Szyperski 2002].

The MApp-IDEA tool is based on components aimed at reusability and interoperability between systems and services. The components were designed to be independent and self-contained, making them highly reusable in different contexts and projects [Clements et al. 2002]. By reusing existing components, we can save time and effort, leading to increased productivity and improved software quality.

We highlight important keys about component-based development adopted by the MApp-IDEA tool:

- **Modularity**. Components are developed, tested, and maintained independently, allowing for better code organization and easier maintenance [Budgen 2003]. Changes made to one component are less likely to have an impact on other parts of the system, making it easier to update and evolve the software.
- **Scalability**. Components are designed to be loosely coupled and independent, so it becomes easier to scale specific parts of a software system without affecting the entire application [Szyperski 2002]. This enables efficient resource utilization and

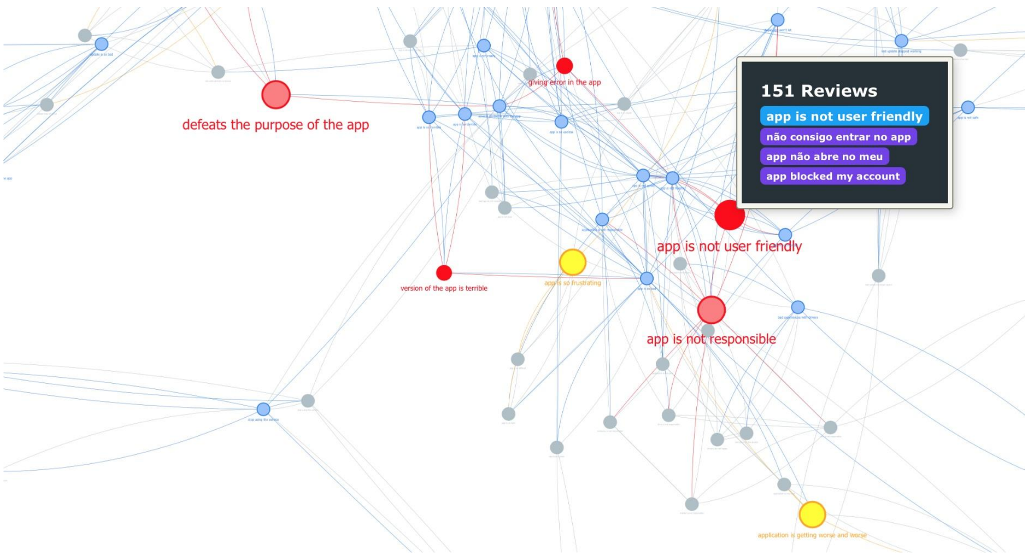


Figure 18. Heat map details. The size and color of each node represent the volume and the critical issue. The larger the node diameter, the greater the number of reviews associated with that issue. Colors vary from light gray to red

ensures optimal performance even under high loads.

- **Collaboration**. By providing well-defined interfaces and contracts, components enable developers to work together effectively [Clements et al. 2002]. Teams can focus on developing specific components (e.g., APIs) that can be seamlessly integrated into the larger system. This encourages specialization and facilitates code reuse and knowledge sharing within the development team.
- **Maintainability**. The modularity of components makes updating or replacing individual components easier without impacting the entire system [Budgen 2003]. This allows for efficient bug fixes, enhancements, and system evolution, making software maintenance less challenging and costly.

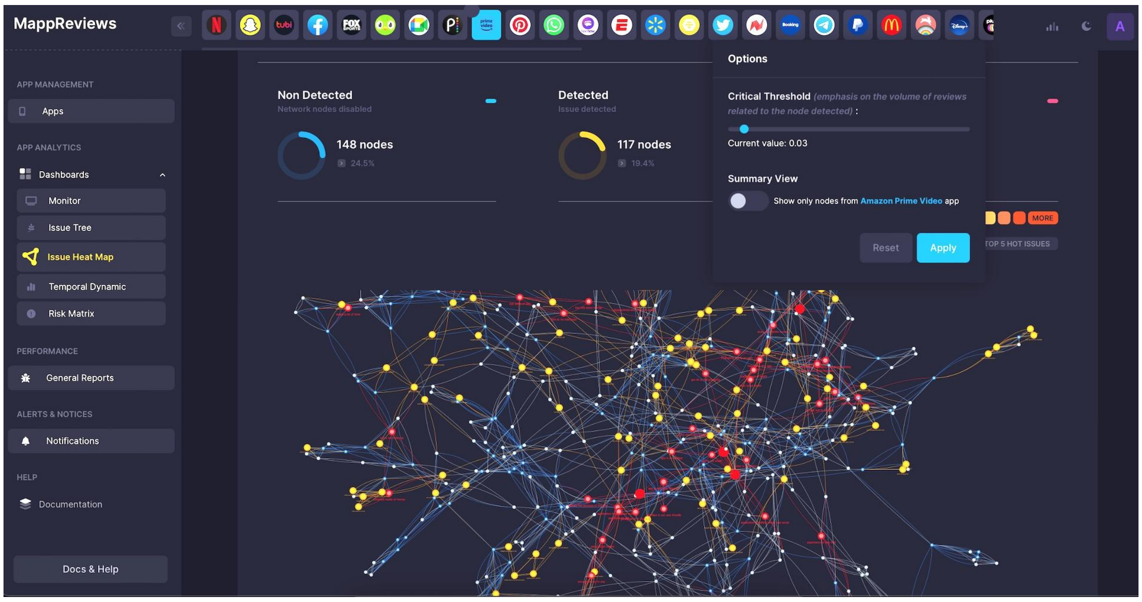
The component-based development approach adopted in the MApp-IDEA tool offers numerous advantages. It promotes reusability, modularity, scalability, collaboration, maintainability, and extensibility. By leveraging pre-built components and assembling them into a cohesive system, we can streamline development and improve productivity.

4.2. Technologies Schema

The development of the MApp-IDEA tool involved utilizing several key technologies, e.g., frameworks, libraries, NL models, and APIs. Each of these technologies played a crucial role in different aspects of the tool, contributing to its functionality and user experience.

To facilitate comprehension of the technology interaction, we have organized it into a stacked layer scheme, where the technologies are grouped based on different scope levels. This division allows for a clearer understanding of the relationships and interactions between the various components involved, as follows:

- **Data collect**. Involves the process of collecting data, e.g., reviews and app metrics.
- **Opinion Mining**. Involves analyzing text to identify and classify issues from opinions expressed by app users.



Options

Critical Threshold *(emphasis on the volume of reviews related to the node detected)* :

Current value: 0.03

Summary View

Show only nodes from Amazon Prime Video app

Reset Apply

Figure 19. Variations in the color scheme of the heat map graph based on the sensitivity threshold

- **Persistence**. Refers to the storage and retrieval of data in a durable and reliable manner.
- **Application**. Refers to the practical implementation and utilization of a system or software. It involves software components, integrating different technologies and user interfaces to provide functionalities and solutions to end users.
- **Synchronization**. Involves data replication, conflict resolution, and communication protocols to achieve consistency and maintain coherence between entities and data.

Figure 28 shows an overview of the technologies separated by the stacked layer scheme. The following is a summary list of technologies. Additional information can be found in the project's public repository on Github [1].

- **Programming languages:** PHP, Python, Javascript, CSS (style), and HTML (markup);
- **Frameworks:** Laminas, Bootstrap;
- **NL Model:** DestilBERT;
- **Database:** Postgres; and
- **Libraries:** Faiss, Nltk, NetworkX, Sklearn, VADER, Transformers, Pandas, Numpy, SciPy, ONNX Runtime, Vijs, Am5Chart, and others.

These technologies stack facilitated seamless server-side and client-side functionality integration, enabling an efficient and user-friendly tool for data analysis.

Several of these technologies intermingle and depend on each other to handle specific functionalities, e.g., crawling, issue detection, and synchronization. For example, PHP code made background calls to Python code, which utilized its rich ecosystem of libraries and tools for these tasks.

[1] `https://github.com/vitormesaque/mapp-idea`

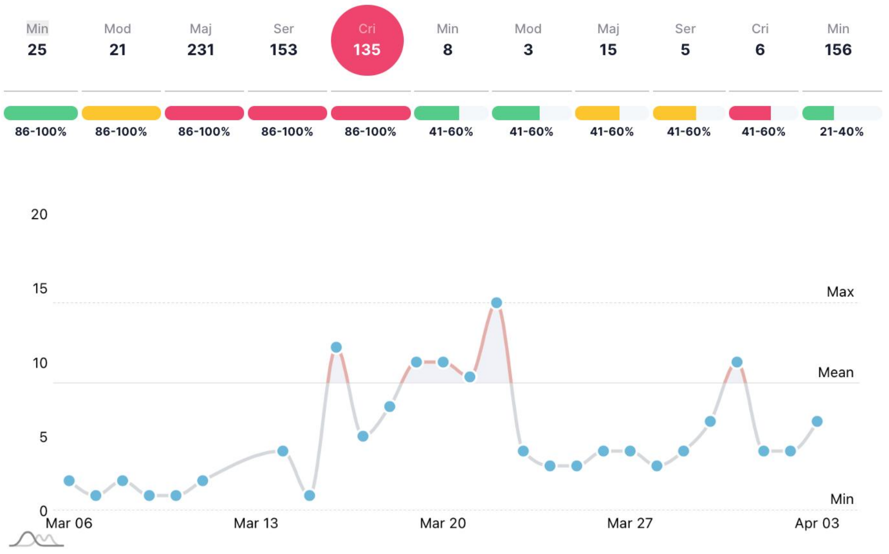
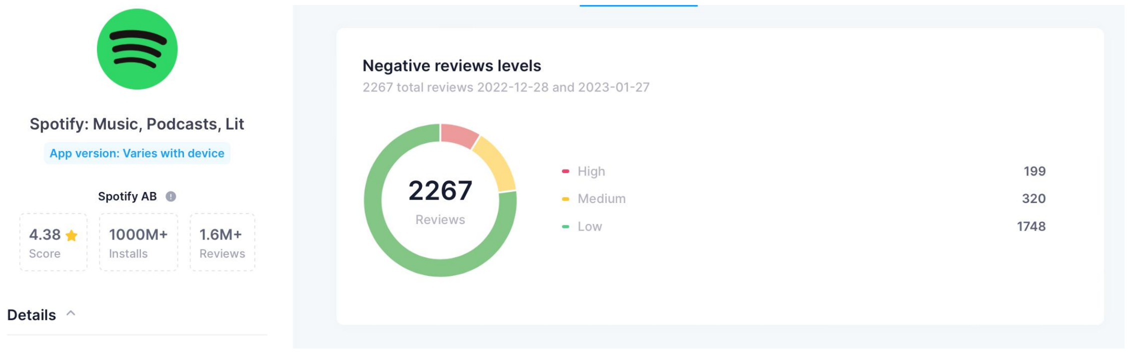


Figure 20. Time evolution of a risk matrix cell


Spotify: Music, Podcasts, Lit

App version: Varies with device

Spotify AB

4.38 Score | 1000M+ Installs | 1.6M+ Reviews

Details

Negative reviews levels

2267 total reviews 2022-12-28 and 2023-01-27

2267 Reviews

High	199
Medium	320
Low	1748

Figure 21. General report of issue classification distribution in 3 critical levels

Through the RESTFul bus, with log systems, notifications, and listeners, it is possible to integrate all technologies transparently for the user. For instance, after completing each step, the Python module responsible for issue detection sends a message to the communication bus (Figure 31). Subsequently, the PHP modules can access the bus as a listener to check for new communication messages.

4.3. Patterns

Design patterns and architectural patterns are both important concepts in software engineering, but they differ in their scope and level of abstraction. The following sections will discuss the patterns incorporated into the MApp-IDEA tool.

4.3.1. Architectural Patterns

Architectural patterns deal with higher-level structures and the organization of entire software systems. They provide a framework for designing the overall structure and interaction between major components and subsystems. Architectural patterns define the funda-

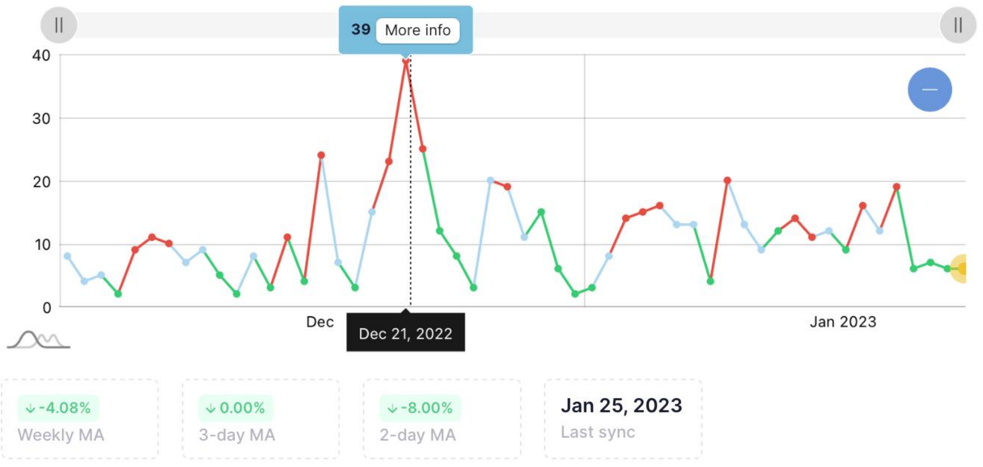


Figure 22. Uptrend detector based on moving average of the time series displayed in three colors on the lines

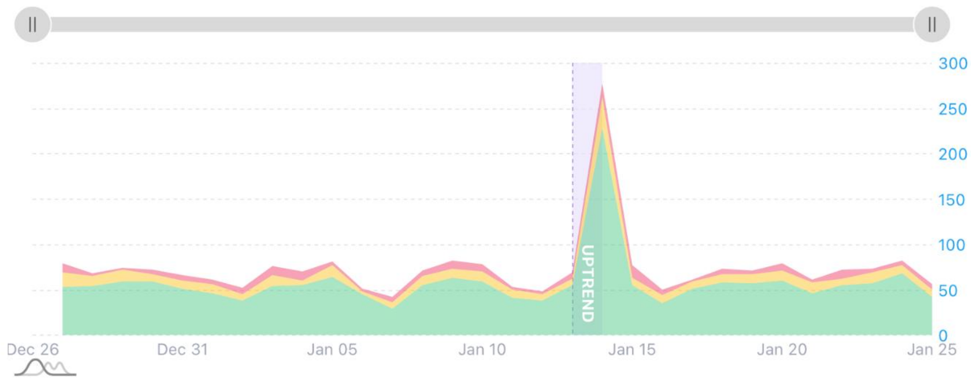


Figure 23. Stacked area chart with three priority levels of issues stacked on each other

mental principles and guidelines for system organization, distribution of responsibilities, and communication between different parts of the system [Sommerville 2013].

Model-View-Controller Pattern – MApp-IDEA tool uses the MVC (Model-View-Controller) architectural pattern widely used in software development. It provides a structured approach for designing and organizing applications by separating the concerns of data, presentation, and user interaction [Gamma et al. 1995].

In the MVC pattern, the application is divided into three main components [Gamma et al. 1995, Reenskaug 1979]:

- **Model** represents the application's data and business logic. It encapsulates data access, data manipulation, and business rules. The Model component is responsible for managing the state and behavior of the application.
- **View** is responsible for the application's presentation layer. It displays the data from the Model to the user and handles the user interface elements. The View is

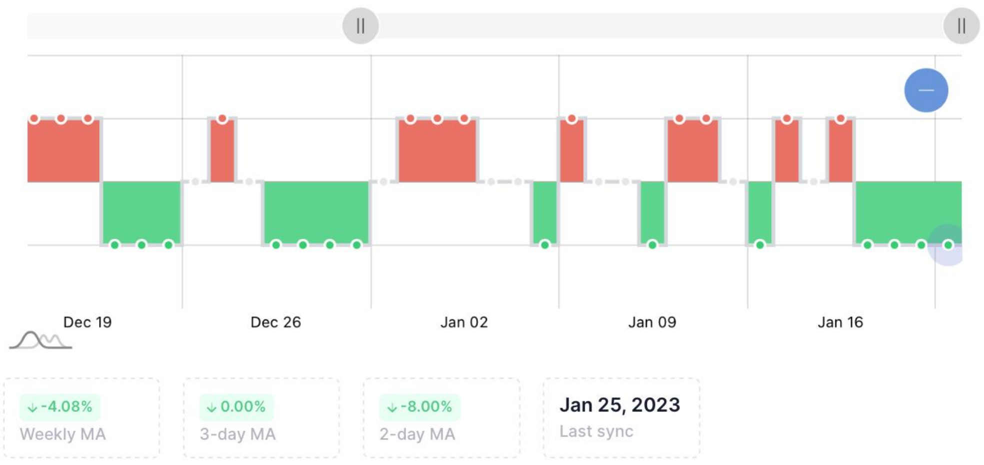
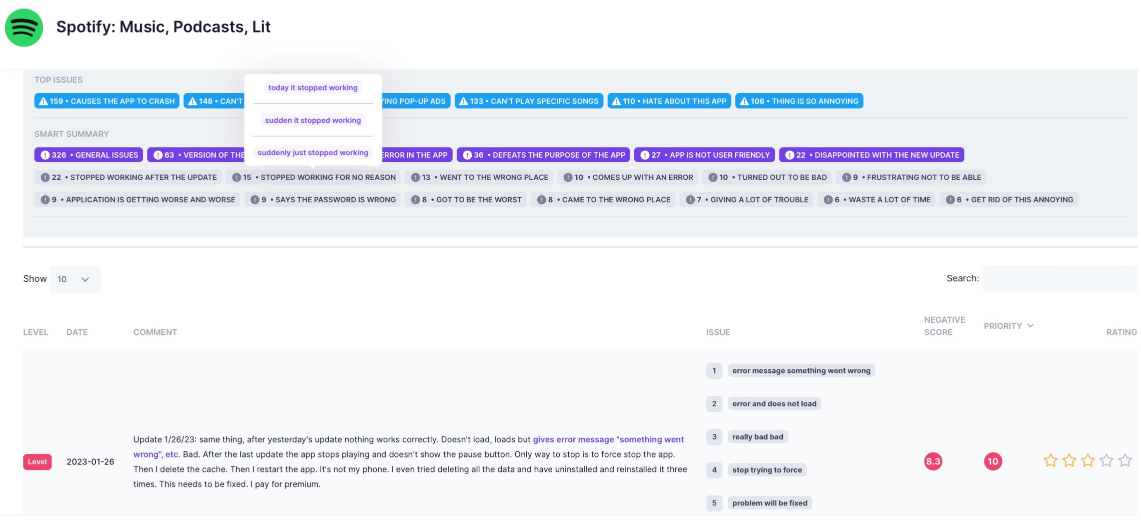


Figure 24. Trend detector to emphasize uptrends and downtrends in the time series



Spotify: Music, Podcasts, Lit

TOP ISSUES

159 • CAUSES THE APP TO CRASH | 148 • CAN'T | ING POP-UP ADS | 133 • CAN'T PLAY SPECIFIC SONGS | 110 • HATE ABOUT THIS APP | 106 • THING IS SO ANNOYING

today it stopped working

sudden it stopped working

suddenly just stopped working

SMART SUMMARY

326 • GENERAL ISSUES | 63 • VERSION OF THE | RROR IN THE APP | 36 • DEFEATS THE PURPOSE OF THE APP | 27 • APP IS NOT USER FRIENDLY | 22 • DISAPPOINTED WITH THE NEW UPDATE

22 • STOPPED WORKING AFTER THE UPDATE | 15 • STOPPED WORKING FOR NO REASON | 13 • WENT TO THE WRONG PLACE | 10 • COMES UP WITH AN ERROR | 10 • TURNED OUT TO BE BAD | 9 • FRUSTRATING NOT TO BE ABLE

9 • APPLICATION IS GETTING WORSE AND WORSE | 9 • SAYS THE PASSWORD IS WRONG | 8 • GOT TO BE THE WORST | 8 • CAME TO THE WRONG PLACE | 7 • GIVING A LOT OF TROUBLE | 6 • WASTE A LOT OF TIME | 6 • GET RID OF THIS ANNOYING

Show 10

Search:

LEVEL	DATE	COMMENT	ISSUE	NEGATIVE SCORE	PRIORITY	RATING
Level	2023-01-26	Update 1/26/23: same thing, after yesterday's update nothing works correctly. Doesn't load, loads but gives error message "something went wrong", etc. Bad. After the last update the app stops playing and doesn't show the pause button. Only way to stop is to force stop the app. Then I delete the cache. Then I restart the app. It's not my phone. I even tried deleting all the data and have uninstalled and reinstalled it three times. This needs to be fixed. I pay for premium.	1 error message something went wrong 2 error and does not load 3 really bad bad 4 stop trying to force 5 problem will be fixed	8.3	10	★★★☆☆

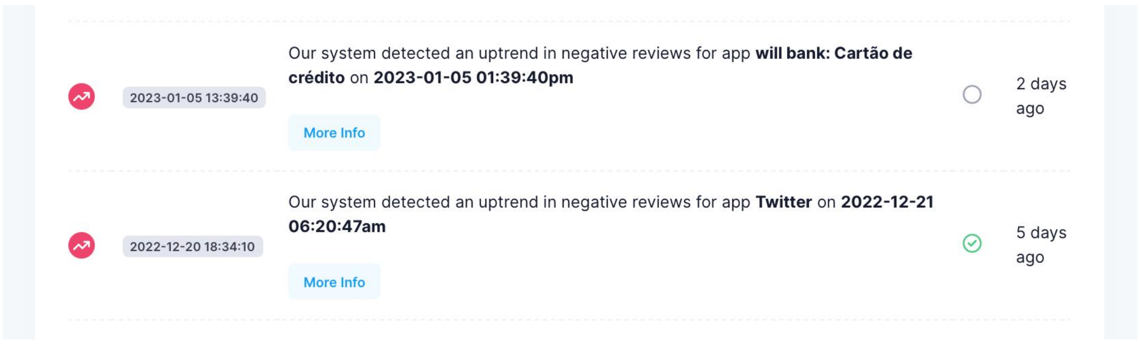
Figure 25. Details of a point in the Spotify app time series.

focused on providing a visually appealing and intuitive user interface.

- **Controller** acts as an intermediary between the Model and the View. It receives input from the user or external systems, performs the necessary actions, and updates the Model and View accordingly. The Controller handles user interactions, interprets requests, and coordinates data flow between the Model and the View.

The MVC pattern promotes modularity, maintainability, and code reusability by separating data, presentation, and user interaction concerns. It allows developers to change one component without affecting the others, facilitating code organization and collaboration in larger projects [Gamma et al. 1995].

Overall, the MVC pattern provides a clear separation of responsibilities, making developing, testing, and maintaining software applications easier. It is widely used in web development frameworks, desktop applications, and mobile app development.


2023-01-05 13:39:40

Our system detected an uptrend in negative reviews for app **will bank: Cartão de crédito** on **2023-01-05 01:39:40pm**

2 days ago

2022-12-20 18:34:10

Our system detected an uptrend in negative reviews for app **Twitter** on **2022-12-21 06:20:47am**

5 days ago

Figure 26. Notification system when there is an upward or downward trend of issues

In our project, we use the Laminas framework [2] to support the implementation of the MVC architectural pattern. The Laminas is a PHP framework for building robust web applications.

One of the key advantages of the Laminas is its modular architecture. The framework is designed as a collection of independent and reusable modules, allowing developers to selectively use and integrate the necessary components for their specific application requirements. This modular approach promotes code organization, reduces development time, and facilitates code reuse and integration across multiple projects.

In addition to its modular architecture, the Laminas offers seamless integration with various databases. It provides convenient abstractions and utilities for working with different database systems, making it easier for developers to handle database operations such as querying, data manipulation, and transaction management. This integration simplifies the development process and ensures compatibility with different database backends.

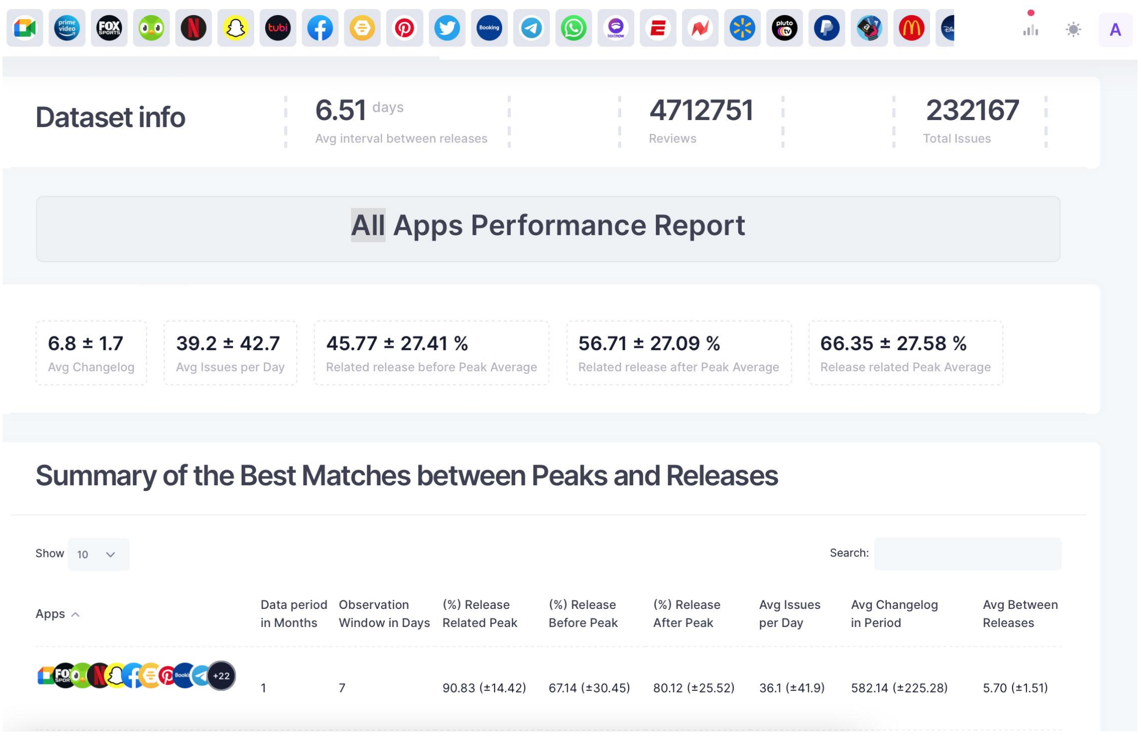
Furthermore, the Laminas Framework emphasizes the importance of community-driven development. It has an active and supportive community that provides regular updates, extensive documentation, and a wide range of resources. This community-driven approach fosters collaboration, knowledge sharing, and continuous improvement, making it easier for developers to learn and leverage the capabilities of the Laminas Framework.

4.3.2. Design Patterns

As described by [Gamma et al. 1995], design patterns focus on solving specific design problems at the class or object level. They provide reusable solutions to recurring design challenges and encapsulate best practices for designing individual components or interactions within a system. Design patterns address object creation, structuring relationships, and behavior delegation.

Several popular design patterns have been widely adopted and studied in the field. We highlight the characteristics and applications of the main patterns (Factory, Singleton, Observer, Adapter, Strategy, and Table Data Gateway) [Gamma et al. 1995, Fowler 2002]

[2] `https://getlaminas.org`



Dataset info

6.51 days
Avg interval between releases

4712751
Reviews

232167
Total Issues

All Apps Performance Report

6.8 ± 1.7
Avg Changelog

39.2 ± 42.7
Avg Issues per Day

45.77 ± 27.41 %
Related release before Peak Average

56.71 ± 27.09 %
Related release after Peak Average

66.35 ± 27.58 %
Release related Peak Average

Summary of the Best Matches between Peaks and Releases

Show 10

Search:

Apps	Data period in Months	Observation Window in Days	(%) Release Related Peak	(%) Release Before Peak	(%) Release After Peak	Avg Issues per Day	Avg Changelog in Period	Avg Between Releases
+22	1	7	90.83 (±14.42)	67.14 (±30.45)	80.12 (±25.52)	36.1 (±41.9)	582.14 (±225.28)	5.70 (±1.51)

Figure 27. Notification system when there is an upward or downward trend of issues

used in the MApp-IDEA tool.

- **Factory**. Focuses on creating objects without specifying their concrete classes. It encapsulates object creation logic within a separate class or method, providing a centralized and flexible approach to object instantiation [Gamma et al. 1995]. The Factory pattern promotes easy modification and extension of the object creation process.
- **Hydrator**. Focuses on separating the process of object creation from the process of populating that object with data. It acts as a bridge between data sources (e.g., databases or APIs), and object-oriented programming models.
- **Singleton**. Restricts the instantiation of a class to a single object. It ensures that only one instance of a class exists throughout the application, providing global access to that instance. The Singleton pattern is useful in scenarios where a single shared resource or state must be accessed from multiple system parts. It simplifies coordination and avoids unnecessary resource duplication.
- **Observer**. Establishes a one-to-many dependency between objects, where the dependent objects (observers) are notified and updated automatically when the state of the subject object changes [Gamma et al. 1995]. This pattern decouples the subjects and observers, enabling them to evolve independently and reducing their interdependencies.
- **Adapter**. Allows incompatible interfaces of different classes to work together. It acts as a bridge between two incompatible interfaces, converting the interface of one class into another that clients expect [Gamma et al. 1995]. The Adapter pattern enables the integration of legacy code or third-party libraries into new systems

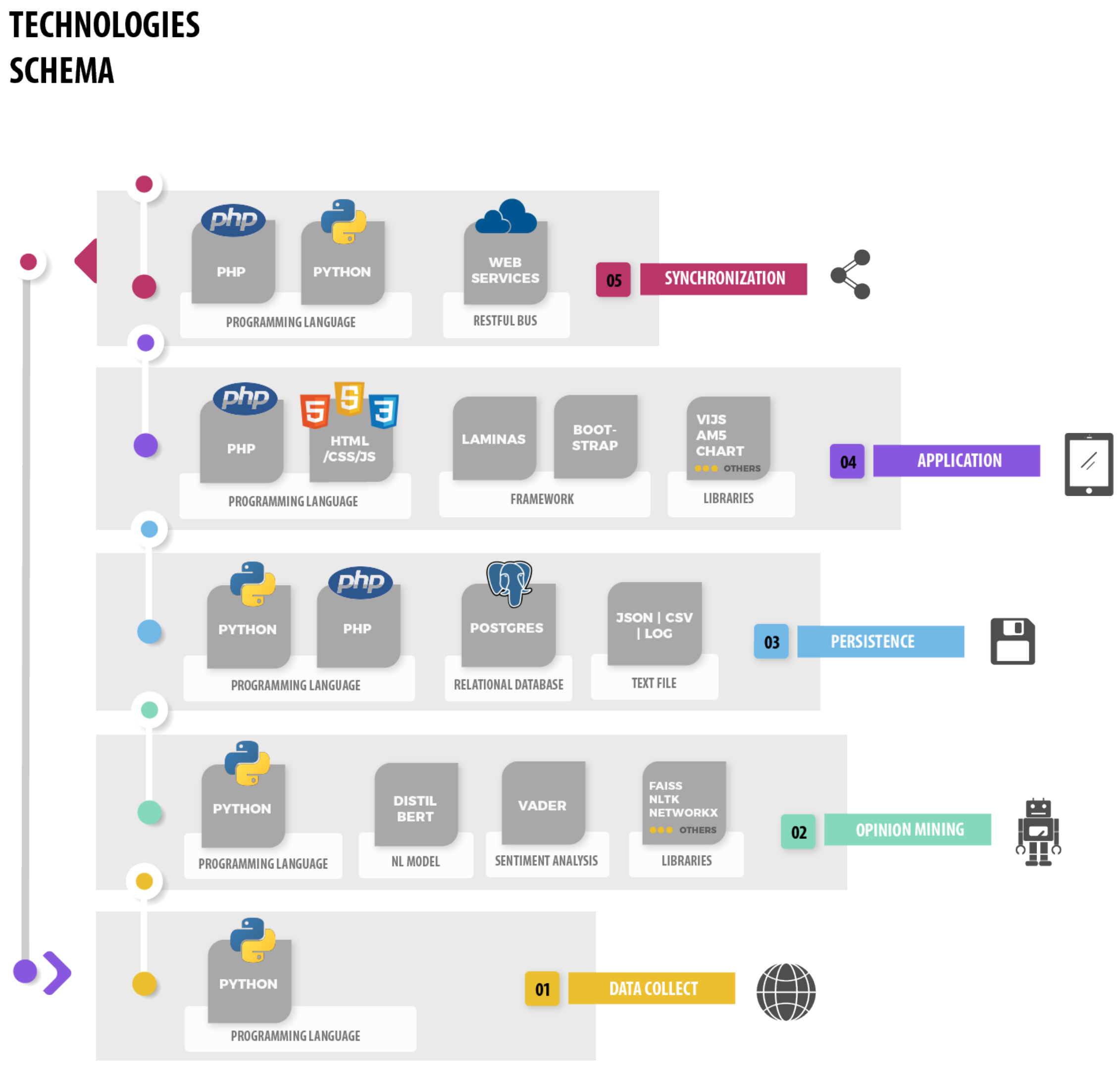


Figure 28. Scheme of technologies used separated by stacked layers

without modifying their existing interfaces, promoting component interoperability.

- **Strategy**. Focuses on encapsulating interchangeable algorithms or behaviors within a family of classes. It allows clients to select different strategies dynamically, depending on their requirements, without modifying the client code [Gamma et al. 1995]. The Strategy pattern promotes flexible design and enables the easy extension and modification of the system's behavior.

- **Table Data Gateway**. Provides a centralized gateway for accessing a database or data source. It encapsulates the data access logic within a single class, which handles all interactions with the underlying data store. The Table Data Gateway pattern abstracts the database operations and provides a convenient interface for clients to query and manipulate data [Fowler 2002].

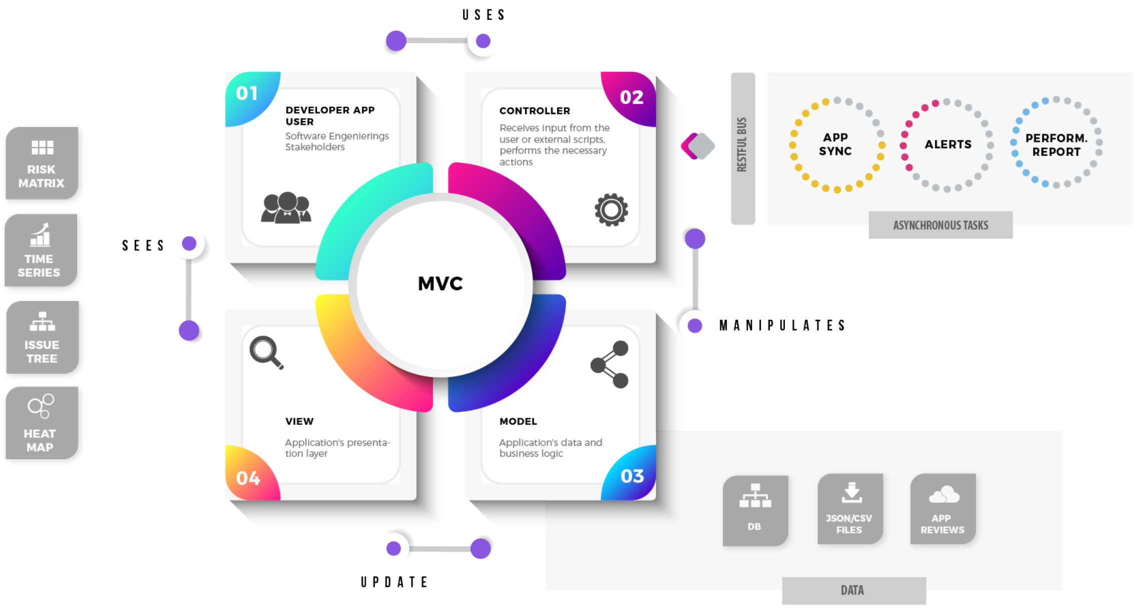


Figure 29. MVC architecture of the MApp-IDEA tool

4.4. Graphical Interface

To build a robust and standardized graphical interface, we use a well-known framework called Bootstrap [3] to build. It provides pre-designed and responsive components, saving time and effort. Bootstrap ensures visual consistency and a professional standard. It has extensive documentation and a supportive community. The framework is mobile-first, enhancing responsiveness and usability.

The MApp-IDEA aims to provide an efficient and user-friendly experience for users accessing it through different devices, including smartphones, tablets, and desktop computers. By implementing a responsive design, the MApp-IDEA tool ensures that users can easily navigate and interact with the tool's features, regardless of the screen size or device they use. This adaptability enhances usability and accessibility, allowing users to conveniently access the tool's functionality on their preferred devices. Additionally, a responsive graphical interface reduces development and maintenance costs by eliminating the need to create separate versions for different devices.

We developed dark mode navigation to improve the navigation experience, as Figure 30 illustrates. The availability of a dark mode option in apps enhances visual comfort, saves battery life, and improves accessibility. Dark mode provides a high-contrast interface that can benefit users with visual impairments or sensitivities. The darker background and lighter text make it easier for individuals with low vision or color blindness to read and navigate the app's content.

4.5. RESTful Bus

The RESTful bus refers to a messaging infrastructure that follows the principles of REST architectural style. The RESTful bus allows communication and coordination between different software components or services using HTTP as the underlying protocol. It

[3] https://getbootstrap.com

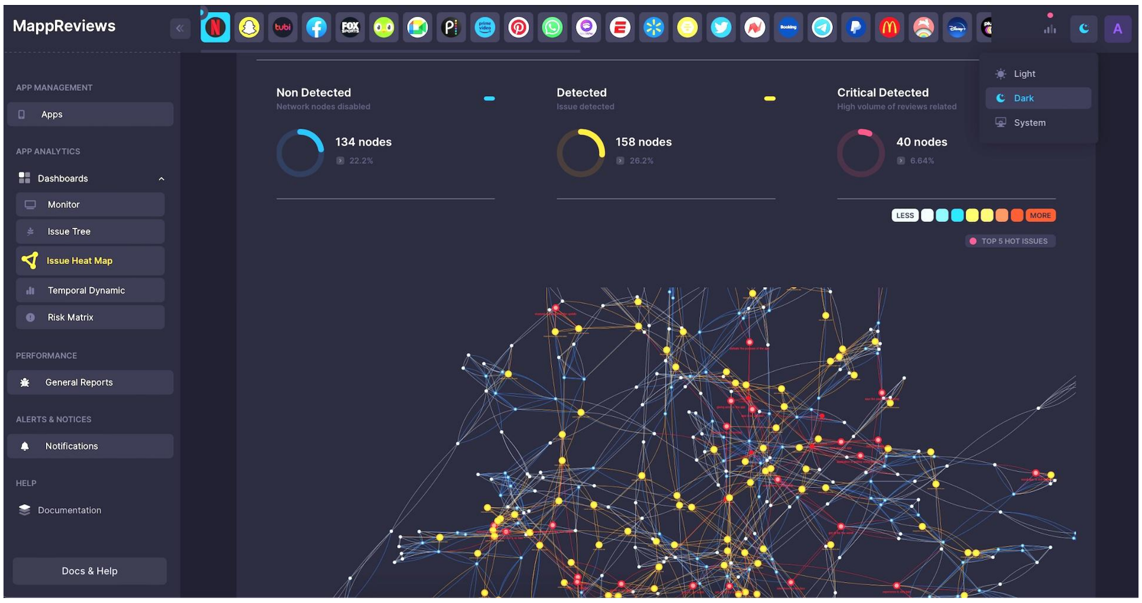


Figure 30. Dark mode home screen

relies on the principles of REST, such as the use of uniform resource identifiers (URIs) to identify resources, stateless communication, and standard HTTP methods (e.g., GET and POST) for performing operations on these resources.

For better understanding, we define communication messages into two distinct levels: i) RESTFull bus messages, and ii) alert trigger messages, as follows:

- **RESTFull bus messages**. Refer to the general communication exchanged between different components or nodes within the bus service.
  - **Notification**. Notifications provide general information about certain events, system states, or updates.
- **Alert system messages**. Messages were sent about monitoring issues and risks associated with trends detected by the tool.
  - **Notice**. Notices are informational messages that communicate general information without requiring immediate attention or action.
  - **Alert**. Alerts are specific messages that are used to indicate urgent or critical situations that require immediate attention.

4.5.1. Synchronization

The MApp-IDEA tool offers a remarkable capability of real-time issue analysis and prioritization. It is essential to download reviews and detect issues in the background to enable this analysis, considering the substantial amount of data being processed and the associated computational demands. Given that many operations are conducted in the background, a communication bus becomes crucial for facilitating the exchange of status messages among the involved components. Also, background services are constantly running with synchronization tasks and alerts.

Once an app is registered in the tool, it is periodically synchronized. The sync period can be customized, but it is five minutes by default. In the scheduled time interval for synchronization, the process is called in the background, making a series of checks to find the last updated review and processing the newly collected reviews. To optimize the review download and processing time, the system estimates how many reviews will be downloaded according to the time of the last update and the average of reviews the app receives per day.

The system performs the following steps in the background on synchronization illustrated in Figure 31.

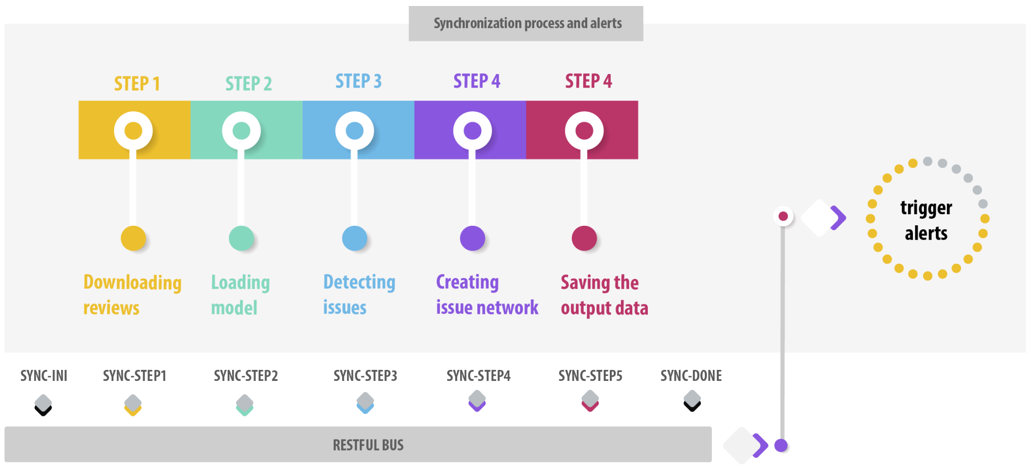


Figure 31. Synchronization process steps and triggering alerts

Depending on the volume of reviews processed, the synchronization may take a few minutes, but the user can follow the processing progress.

During the synchronization process, whenever there is a status update, a message is sent to the RESTful bus to notify about the new status. This enables the view layer to query the bus and retrieve the updated status information.

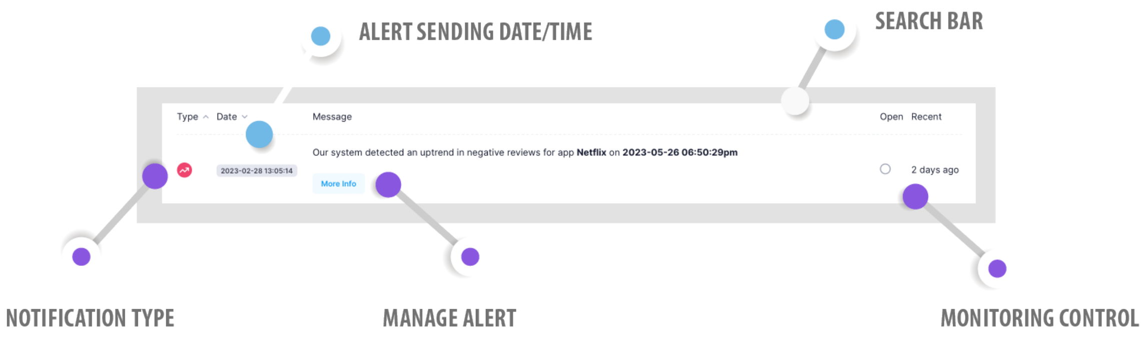
4.5.2. Alert Trigger

Once the synchronization process is completed, the system notifies the RESTful bus about the synchronization status. Subsequently, the RESTful bus forwards this information to the alert trigger system, which initiates the scanning process to identify and trigger alerts and notices related to app issues and risks. Similar to the synchronization process, the alert trigger system operates in real-time.

By utilizing the RESTful bus, we ensure that potential app issues and risks are promptly detected, and appropriate actions are taken through efficient communication and real-time availability of status updates, enhancing the overall responsiveness and functionality of the application.

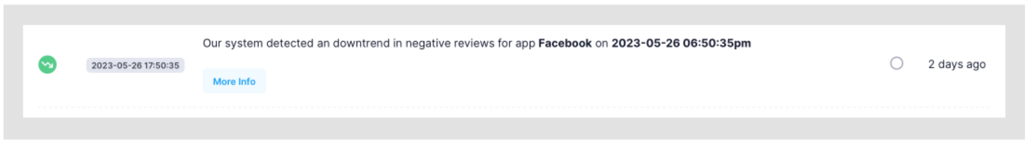
As an illustration, Figure 32 displays an Alert message dated May 26, generated by the Netflix app to inform users about an app update that revised the policy regarding simultaneous screens in the family plan. Figure 33 exhibits a notice message from the Facebook application on May 25, 2023.

The MApp-IDEA tool includes a dedicated component for message management,



2023-02-28 13:05:14 Our system detected an uptrend in negative reviews for app **Netflix** on **2023-05-26 06:50:29pm** 2 days ago

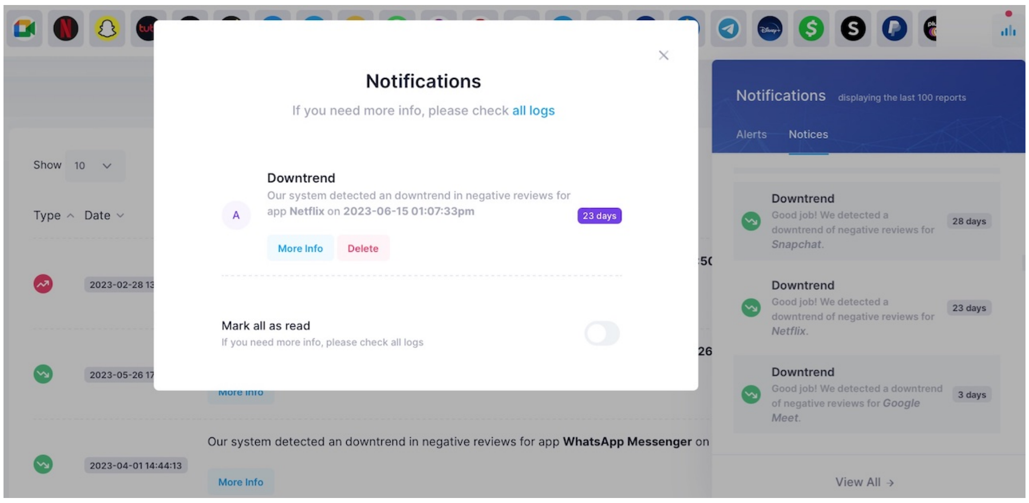
Figure 32. Alert message from the Netflix app on May 26, when there was an app update changing the policy for simultaneous screens in the family plan.



2023-05-26 17:50:35 Our system detected an downtrend in negative reviews for app **Facebook** on **2023-05-26 06:50:35pm** 2 days ago

Figure 33. Facebook application notice message on May 25, 2023

ensuring that all sent messages are stored, as shown in Figure 34. When a message is opened, the system marks it as read to provide users with a clear indication of their message status. Even if a user chooses to delete a message (whether it's an alert or a notice), the system retains a record of it in the database for traceability purposes. Therefore, while the message may appear unavailable to the user, it is not permanently deleted from the system. This approach ensures data integrity and allows tracking and auditing of message-related activities.



Notifications

If you need more info, please check all logs

Downtrend
Our system detected an downtrend in negative reviews for app **Netflix** on **2023-06-15 01:07:33pm** 23 days

Mark all as read
If you need more info, please check all logs

Notifications displaying the last 100 reports

Downtrend
Good job! We detected a downtrend of negative reviews for *Snapchat*. 28 days

Downtrend
Good job! We detected a downtrend of negative reviews for *Netflix*. 23 days

Downtrend
Good job! We detected a downtrend of negative reviews for *Google Meet*. 3 days

2023-04-01 14:44:13 Our system detected an downtrend in negative reviews for app **WhatsApp Messenger** on

Figure 34. Component for managing triggered messages related to the detection and prioritization of issues.

5. Concluding Remarks

We discussed the design and architecture of the MApp-IDEA tool, focusing on key aspects such as component-based development, technologies schema, patterns, and the graphical interface.

The design and architecture of the MApp-IDEA tool were carefully crafted to deliver a powerful and user-friendly data analysis solution. Reusable components were leveraged through component-based development, enabling modularity, scalability, collaboration, and maintainability. Various technologies, such as frameworks, models, and libraries, were seamlessly integrated to create a cohesive system. Architectural patterns (e.g., MVC), and design patterns (e.g., Factory, Singleton, Adapter, Strategy, and Table Data Gateway), enhanced code structure, flexibility, and extensibility. The graphical interface, built with Bootstrap, ensured a responsive and visually appealing user experience. The synchronization process and real-time alerts optimized the processing of app reviews. In summary, the MApp-IDEA tool embodies industry best practices, resulting in a powerful and user-friendly solution for analytical data exploration.

Source code files and a video demonstration that support the findings and contributions are available at:

- Source code:`https://github.com/vitormesaque/mapp-idea`
- Demonstration: `https://youtu.be/VPHZ8LtjoJo`

References

Araujo, A., Golo, M., Viana, B., Sanches, F., Romero, R., and Marcacini, R. (2020). From bag-of-words to pre-trained neural language models: Improving automatic classification of app reviews for requirements engineering. In *Anais do XVII Encontro Nacional de Inteligência Artificial e Computacional*, pages 378–389. SBC.

Araujo, A. F., Gôlo, M. P., and Marcacini, R. M. (2022). Opinion mining for app reviews: an analysis of textual representation and predictive models. *Automated Software Engineering*, 29(1):1–30.

Budgen, D. (2003). *Software Design*. Addison-Wesley Professional.

Chatfield, C. (2003). *The analysis of time series: an introduction*. Chapman and hall/CRC.

Clements, P., Bass, L., and Kazman, R. (2002). *Software Architecture in Practice*. Addison-Wesley Professional.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Hutto, C. and Gilbert, E. (2014). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the international AAAI conference on web and social media*, volume 8, pages 216–225.

Johnson, J., Douze, M., and Jégou, H. (2017). Billion-scale similarity search with gpus.

Lima, V., Araújo, A., and Marcacini, R. (2022). Temporal dynamics of requirements engineering from mobile app reviews. *PeerJ Computer Science*, 8:e874.

MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

Mihalcea, R. and Radev, D. (2011). *Graph-based natural language processing and information retrieval*. Cambridge university press.

Reenskaug, T. (1979). Models-views-controllers. Retrieved from `https://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html`.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Sommerville, I. (2013). *Engenharia de Software*. Pearson Education, [S.l.], 9th edition.

Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Professional.

Williamson, E. (2010). *Lists, Decisions and Graphs*. S. Gill Williamson.