Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 238 (2024) 208–215

Procedia Computer Science

www.elsevier.com/locate/procedia

The 15th International Conference on Ambient Systems, Networks and Technologies (ANT)
April 23-25, 2024, Hasselt, Belgium

Design of a mobile app for crisis guidance

Kimmo Kiiveri[a,*], Jyrki Liesivuori[b], Seppo Virtanen[a], Jouni Isoaho[a]

[a]*Department of Computing, University of Turku, 20114 Turku, Finland*
[b]*Department of Biomedicine, University of Turku, 20114 Turku, Finland*

Abstract

This study examines the use of an Online Application Generator (OAG) for low code mobile app design and development, focusing on transforming crisis management guidance from PDF format to a published platform-independent mobile app. We compare the level of difficulty experienced by a citizen developer and a full-stack developer, considering ease of development, customization, security, and architecture. Despite OAGs' effectiveness in managing vulnerabilities, they pose risks such as undisclosed tracking and compliance with privacy laws. The approach was found feasible for citizen developers in terms of rapid mobile app development although we observed limitations in localization and in preserving privacy.

Keywords: Mobile application; LCDP; OAG; Citizen developer; Crisis guidance

* Corresponding author.
E-mail address: kimmo.kiiveri@utu.fi

10.1016/j.procs.2024.06.017

1. Introduction

In the field of medicine, the utilization of Personal Digital Assistants (PDAs) for guidance was adopted at an early stage [1,2], and the results for web-based guidance have been positive [3,4]. One of the earliest examples outside medical use is the Work Environment Profile survey [5], which used a PDA device with custom software to conduct a survey. In education, mobile apps are widely used [6]. The International Labor Organization has a series of mobile apps intended for improving occupational safety and health at workplaces, for ergonomics and stress prevention [7] and the Estonian Naiskodukaitse [8] (Women's voluntary defense organization) released the 'Be prepared!' mobile app for preparedness. Mobile apps are indisputably essential across various domains.

The process of developing a mobile app can be costly and necessitates specialised knowledge of the technology. Low-code development platforms (LCDPs) such as online application generators (OAGs) have several advantages. First and foremost, using an OAG allows for rapid application development and delivery [9,10]. The OAGs also enable development without extensive experience in programming [11] whereas a full-stack technological approach requires mastering multiple skills from the developer. In today's software development, security and privacy are critical factors that cannot be ignored, but it is not always clear how they are accomplished behind the scenes with OAGs [12].

We investigate how the guidance for SMEs in various crisis situations is transformed into a mobile app from PDF format through the utilization of an OAG. We empirically analyze and compare the level of difficulty experienced in the development process by two distinct developer profiles: a citizen developer utilizing the OAG, and a full-stack developer employing full-stack technologies. This comparative analysis seeks to elucidate the efficiency and potential limitations of OAGs in the context of crisis management application development for SMEs. To our knowledge, no previous studies on the matter exist and this study aims to fill the gap.

2. Research Methodology

We define a citizen developer as a person without programming experience but with an understanding of spreadsheet logic and the capability of logical problem solving. A full-stack developer is defined as a person who can develop both client and server-side software, mastering browser, server, and database development. Full-stack technology was chosen as a point of comparison because most of the OAGs produce Progressive Web Applications (PWAs). A PWA is essentially a web page that can also be published in an app store. Implementation as a PWA provides the advantage of operating system independence across various mobile platforms. However, this advantage is somewhat mitigated by the differences in how various browsers implement PWAs.

In selecting a development platform for our study, key considerations included the simplicity of the development process and the capability to support a unified app version compatible across all major operating systems, rather than necessitating separate versions for Android and iOS. Priority was given to platforms that obviated the need for intricate database setup and ongoing maintenance. The selection included various low-code development environments, which are notable for their rapid evolution. For instance, Google AppMaker was experimented with, but the product was discontinued during our selection process.

Out of 21 platforms analyzed by Luo et al. [13], we initially focused on four (Adalo, AppGyver, Thunkable and Glide) and outside of that study, on Oracle Application Express (APEX). APEX was found to be inappropriate for novice citizen developers due to its prerequisite of database knowledge, whereas AppGyver necessitated the integration of an external database. Our final selections were Adalo, Thunkable, and Glide, with the latter two using Google Sheets for data storage. Table 1 summarizes the key features of these platforms. Glide publishes PWAs and has Google Analytics for later usage analysis, which swayed the decision to it. Glide has been part of the citizen developer movement since 2019 [14] and it offers a supportive community [15] for development questions.

Table 1. Table of features considered in OAG selection in December 2020

Feature	Adalo	Thunkable	Glide
UI options	Most	Less	Less
App Store/Play Store Publishing	Yes	Yes	No
PWA App	Yes	No	Yes
User Privacy	Good	Medium	Medium
Analytics	MixPanel	No	Google Analytics
Data Storage	Proprietary	Google Sheets or Airtable	Google Sheets or Glide Tables

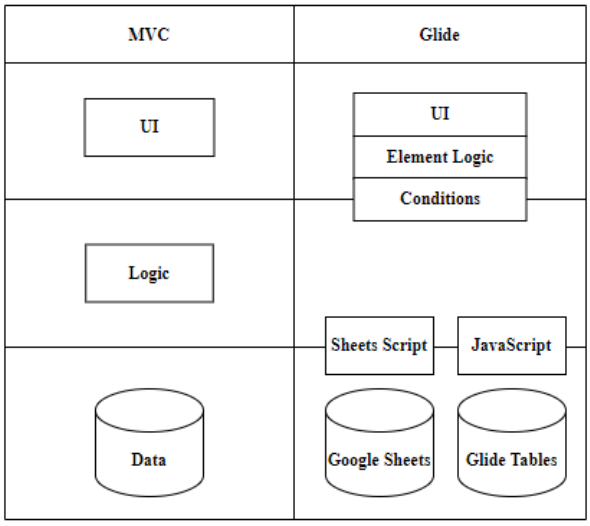
3. Features and implementation of Mobile Crisis Assistant

3.1. Application architecture

Glide adopts the Model-View-Controller (MVC) design pattern, simplifying it into two layers and enabling control logic directly within the UI elements. Due to the limitations of this approach, there are more options for control logic, consequently introducing complexity into the architecture. In Glide the logic can reside in the UI element or its Conditions extension, in Google Sheets script, or in Glide Tables with Javascript. The architectural differences are shown in Figure 1.

The application created using Glide is a Progressive Web Application (PWA), not a native mobile app. This approach simplifies development and distribution, as it only requires a URL for sharing rather than submission to an app store. However, we found that Glide's PWA did not fully exploit all PWA capabilities. These development features highlight the inherent technical limitations of using an OAG like Glide compared to a full-stack development approach, which typically offers more flexibility in implementing complex features.

Fig. 1. Architectural differences between the MVC pattern and the Glide implementation.



3.2. User Interface

Development in Glide, like in other OAGs, is primarily drag-and-dropping of UI elements and binding them to Google Sheets columns. The existing visual appearance is sufficient for most cases and partly customisable. Some

Glide developers had discovered a way to insert custom functionality, much like hacking other platforms or mobile phones. A developer could add JavaScript and CSS to customise the Glide UI element, but since code level modification is not officially allowed, there is always the risk that the next platform update would break it, and this happened when Glide did refactor the UI elements.

During the implementation, Glide introduced the Conditions feature. That enhances the UI element with basic IF-THEN logic. As a result, it is possible to base elementary logic in the UI element on the contents of one or more of the element's fields. As with all coding, the rules must be written carefully. Creating one rule in the drop-down box is simple but creating rules that are more complex requires connecting boxes using AND or OR operations, which requires the developer to use rigorous logic or otherwise maintenance is endangered.

An OAG's UI has fewer functionality and customization options but from a process standpoint this also expedites development. With an OAG there is no UI fatigue due to limitations. The infamous 'pixel tweaking', as with the full-stack approach, just cannot be done. Considering the amount of work needed to make a polished UI is favourable for an OAG, a full-stack developer would require hours to get the pixels right, even if using as much existing code as possible.

3.3. Localization

One of the shortcomings of Glide is the inadequate support for content localization. Standard elements like buttons can be translated in Glide, but there is no possibility to select the content text language based on the user's selection. The ability to translate the program into additional languages would have been useful but doing so currently necessitates creating new apps for each language. In full-stack technologies, localization is mostly a built-in feature, and it is more of a question of enabling it. However, localization might require some content modifications as well since procedures for crisis situations differ in each country. The same methods of guidance might not be effective in all societies.

3.4. RSS Feed

In the app, the News tab has Really Simple Syndication (RSS) feeds from different authorities. This is a definite addition to a static PDF format. RSS updates the information when the configured page has new information. Getting that can be accomplished in one of two ways. One way is to use a web application integration tool like Zapier. In this study, it was considered as an additional tool, and another approach was selected to keep the implementation simpler with fewer tools. In Google Sheets, there is an IMPORTFEED function that fetches the current RSS feed and adds it to Sheets. A separate refresh timer was added with a Google script to update the feed every hour. Glide reacts to changes and then shows them in the app screen, and the news from different authorities can be seen. Glide's update cycle is at the shortest 3 minutes, so it is relatively real time information but cannot be fully trusted. With a full stack approach, that could be more precisely controlled if a specific interval is required. Since the IMPORTFEED approach involves more technical expertise than merely adding an RSS URL to an element, it is debatable whether a citizen developer would be able to integrate the RSS feed.

3.5. App notifications

The PWA approach has one drawback compared to native mobile implementations in that mobile push notifications cannot be used at the operating system level on IOS or Android. There is a possibility to circumvent it, but that would require scripting in Google Sheets and sending messages through email gateways to the SMS service, which is also carrier specific and country specific. An SMS is also more invasive than a mobile app notification. Therefore, the notifications were not implemented.

A full-stack developer has the capability to implement notifications using the W3C Badging API [16] although it is not yet included the W3C's standards track. In conjunction with the Web Periodic Background Synchronization [17] it is possible to update the badge state (notification) at regular intervals during periods when the PWA is not actively used by the user. This adds some privacy concerns too. For example, user location could be tracked with a

malicious implementation or background sync request history could leak to other middleboxes. In addition, it adds the security risks of cryptocurrency mining, Distributed Denial of Service (DDoS), and online ads' fake clicking.

3.6. Data storage

It is very straightforward to start using Google Sheets for data storage, but it can become complicated. A developer can also use Sheets scripts for more complex behaviour if the functionality with Glide is thought out. Additionally, developers can use Glide columns for storage between in addition to Google Sheets. Creating logic is possible using Glide columns and javascript, but it is important to consider if it makes sense to store data there in addition to Google Sheets. Transferring data to another platform is more challenging if it is partially resides kept in Glide Tables and Google Sheets. Long-term development and maintenance risks arise from having data and logic in many separate places. If all the locations where logic is applied are unknown when one of them is altered, it might result in unanticipated consequences. It is difficult to determine which layers affect the data. With full-stack technology, the developer can trace the data and see where changes occur, but that is not possible with an OAG.

3.7. User privacy

In Glide complex UI screen elements need a user id for the row owner, and the user's email address is used as the user id. The email is encrypted as per Glide, but no further information is provided regarding the encryption process. During implementation, a setting was added that made it possible to conceal the email address from the developer's view. This is satisfactory for a private app used by individuals, but not for an organisation in the EU. In the EU, General Data Protection Regulation (GDPR) legislation [18] needs to be taken into account. If the data is stored outside of the EU, the developer needs to define the data transfer mechanism (as in a contract, not technically). Unknown storage has an impact on the end user's possibility to request data correction and erasure. With Glide it is not possible to ensure where the data is stored. Glide as a company for the developers fulfils the GDPR legislation, but that cannot be extended to the apps. Glide highlights that restricting data visibility or using conditions in the UI elements are not security features, but a GDPR compliant implementation is not advised for citizen developers.

Using Google Sheets to store personal data has a similar privacy issue. However, the storage location can be restricted to the EU and, if necessary, a user profile can be managed using Google Workspace (formerly G-Suite). Constructing a user profile management is not a task that a citizen developer could easily do, if at all. In this study, no user management was made to minimize the data collection.

3.8. App security

In the OWASP Top 10 Security Awareness document for developer and web application security [19] there is a new entry from 2021 that is relevant to this study. In fourth place is A04:2021-Insecure Design. That is a new category for 2021, with a focus on risks related to design flaws. More use of threat modelling, secure design patterns and principles, and reference architectures must be used to prevent it. With an OAG, the design is predetermined from the beginning, preventing the citizen developer from making poor decisions affecting architecture.

OAGs handle OWASP 2021 security risks, but full-stack developers need to pay attention to them, especially to A06:2021-Vulnerable and Outdated Components. According to a study [20], it was concluded that half of the vulnerabilities in open-source software needed two years to be disclosed and four years to be fixed.

The PWA is one of the safest methods to implement software. However, no technology, not even the PWA, can ensure complete security. PWAs may have the appearance of new technology since they generate native-like mobile app experiences, but they are primarily enriched web applications with the same security flaws, some of them being due to design. As a result, PWAs are susceptible to all recognized types of online attacks [21].

Glide uses one-time multifactor authentication (MFA) when the app is opened. After the email address is provided, a one-time pin code for signing in is sent to the given email address. During usage, the PIN is not requested again unless the user logs out. Although the PWA uses HTTPS, cookie and session hijacking are one of the most common risks. The PWA stores the authentication cookie of the user for the session. If the user does not explicitly log out, the authentication cookie stays in session and can be hijacked for a certain time. The time depends

on how long the server side keeps the session open before closing it or from the cookie validity time. Glide uses persistent cookies for user recognition.

4. Results

We discovered that, in appropriate cases, an OAG is a rapid way to implement applications. However, there is a significantly greater need for trade-offs between requirements and technological capabilities. The work amount and complexity are less for the OAG development, but that is due to OAG's technological limitations. That also affects the development process in such a way that the iteration from the user feedback cannot refine the implementation since many features are limited. Transforming the guidance from a static PDF format to a mobile application in this study was a successful example. The mobile app, Crisis Assistant, had over 300 downloads and received favorable reception.

The main difference with OAGs to web development is that little prior experience is needed beforehand and most of the skills needed can be learnt as you go, or the OAG even instructs you. Therefore, the first step that needs to be taken is minimal compared to starting any coding, let alone full-stack development. Otherwise, good coding practices and logical thinking apply in both cases.

We estimated the effort needed from a citizen developer and full-stack developer point of view. The areas estimated were User Interface, Localization, RSS Feed, App Notifications, Data Storage, User Privacy, App Security, Testing, Publishing, and Maintenance.

The results are summarized in table 2 and presented in detail in the following.

- **UI design.** Most of the users did not notice any difference between the PWA app made with Glide and native apps. That highlights the point, that the user interface is mostly good enough out of the box, so the time and effort needed for graphical design is less. Therefore, when the OAG UI can fulfil the requirements, we can consider it easy versus medium effort in full-stack development.
- **Localization.** This was one of the shortcomings of the OAG used in the study. Button descriptions are translated automatically by the platform, although only based on the mobile device's language. Translating content would have required building a user management system and proved to be impossible due to user privacy issues. With the full-stack approach, localization is not a problem.
- **RSS Feed.** It is possible to add the RSS Feed to the mobile app, but the addition requires usage of a special function in Sheets with Glide. That was considered hard for citizen developer. In full-stack development, RSS Feed integrations are easily available.
- **App notifications.** Notifications are a mobile operating system's specific feature and adding them to a PWA is not feasible with an OAG. It might be possible with full-stack development, but until there is a standardised implementation, the problem will be challenging to resolve.
- **Data storage.** This was a more complicated issue than originally thought. Initially, it was assumed that the spreadsheet approach would be easier versus the database, but multiple OAG options and the GDPR legislation made it tricky. We still consider it easy with the OAG if the requirements can be fulfilled. Full-stack developers can easily duplicate a database installation, but nevertheless, it requires more work.
- **User privacy.** Due to the legislation in the EU, regarding the ePrivacy Directive and the GDPR, this became the hardest feature to satisfactorily implement. It became apparent that no easy solutions existed and the US based OAG in the study did not seem to have an interest in solving it on an architectural level. With full-stack technologies, privacy is more easily solved, but requires attention and possibly legal consultation.
- **App security.** The study's OAG automatically ensured security through the platform, given PWAs' inherent security. Full-stack technologies can be made more secure, but more technologies and solutions offer more attack vectors, with some of the technologies being open source without guaranteed patching schedules.
- **Testing.** With OAGs, trying and testing is part of the development process, but unit testing or testing the complete app as a stable separate environment is not possible. Currently, there is no mechanism to automatize testing. With the OAG, the testing is the citizen coder's responsibility during the development.

- **Publishing.** Publishing a PWA with the OAG is very easy but there is no versioning possibility. Publishing a native app through an application store or setting up a website for a PWA with a full-stack approach takes considerably more effort. On the other hand, a full-stack developer has more options for the roll-back.
- **Maintenance.** All software needs maintenance during its lifetime, but the possibilities with the OAG might be limited. It would probably be easier to create a new app if the requirements change or the development platform has changes in the behavior. With a full-stack approach, maintenance effort varies based on the original architecture selected, but we considered it to be more than with an OAG.

Table 2. Table of features and the complexity for the developer in an OAG or in the full-stack environment

Feature	OAG	Full-stack
UI Design	Easy	Medium
Localization	Hard	Easy
RSS Feed	Hard	Easy
App Notifications	N/A	Hard
Data Storage	Easy	Medium
User Privacy	Hard	Medium
App Security	Easy	Hard
Testing	Hard	Medium
Publishing	Easy	Medium
Maintenance	Easy	Medium

5. Discussion

Luo et al. [1] observed the three main benefits for low-code development: faster development, ease of study and use, and lower IT costs. These are the results this study achieved as well, although ease of study and use is very dependent on the individual developer´s skills, as we can see from the main challenges. According to Luo, the three main limitations and challenges were the high learning curve, high pricing, and lack of customization. The high learning curve and pricing are contradictory to the benefits but can be explained by the vast user base and versatile use cases. High pricing is dependent on the vendor and their pricing policy. Compared to custom-fit software projects' prices all OAGs are cheap, but for the citizen developer, a small charge might be considered high. Customization is a limitation this study found as well. On the other hand, fewer options expedite the development.

OAGs are a secure way to implement apps for a citizen developer. If a vulnerability exists, it is likely that the OAG provider will address it more effectively and quickly than a full-stack developer or a third-party library, much less a citizen developer. Therefore, one less task with OAGs is security maintenance and updating. However, the risk exists that centralized, and platform wide OAG updates could break something in the existing apps.

A study [12] found that some OAGs might perform tracking in the background without explicitly revealing it. Glide, however, states it in the privacy policy. Even though it would be nearly impossible for a citizen developer to find it, the responsibility still exists.

The OAG may convey to the citizen developer the mistaken impression that everything is in order, despite the responsibility belonging to the developer to adhere to laws that are in effect in the particular nation, especially in the EU. The privacy issues caused two features to be abandoned from the final release of the app: an action diary and a personal favorites task list. They would have been used to store information about what had been done and which actions were found interesting, not a direct violation of GDPR but a privacy issue.

6. Conclusion

This study has critically examined the use of OAGs in mobile app development, focusing on the transformation of crisis management guidance from static PDF format into a dynamic mobile application. The comparative analysis between OAGs and traditional full-stack development methodologies revealed that OAGs offer significant benefits

in terms of rapid development and reduced complexity, particularly for citizen developers without extensive programming experience. OAGs simplify the development process by minimizing the need for in-depth knowledge of multiple programming languages, frameworks, and platforms, typically required in full-stack development.

Our findings indicate that OAGs potentially offer enhanced security compared to full-stack development. The more streamlined approach of OAGs can mitigate the risks, albeit at the expense of reduced customization and functionality. However, it is crucial to note that the responsibility for ensuring compliance with privacy laws and addressing vulnerabilities still rests with the developer.

Future investigations into the efficacy of privacy measures in OAGs will further enhance our understanding in this dynamic field. The incorporation of AI into mobile app development could revolutionize the way apps are created, potentially making complex tasks like localization possible. However, ensuring the quality of AI-generated content and functionality remains a critical consideration.

7. References

[1] Lu, Y. C., Xiao, Y., Sears, A., J. A. Jacko, J. A., 2005. A review and a framework of handheld computer adoption in healthcare. *International Journal of Medical Informatics* 74(5), 409–422. https://doi.org/10.1016/j.ijmedinf.2005.03.001

[2] Malter, D., Davis, T., 2003. Data collection and "real-time" learning using handheld computers. *Applied Occupational and Environmental Hygiene* 18(5), 321–30. https://doi.org/10.1080/10473220301362

[3] Bochicchio, G. V., Smit, P. A., Moore, R., Bochicchio, K., Auwaerter, P., Johnson, S. B., Scalea, T., 2006. Pilot study of a web-based antibiotic decision management guide. *Journal of the American College of Surgeons* 202(3), 459-467. https://doi.org/10.1016/j.jamcollsurg.2005.11.010

[4] Lapinsky, S. E., Wax, R., Showalter, R., Martinez-Motta, J. C., Hallett, D., Mehta, S., Burry, L., Stewart, T. E., 2004. Prospective evaluation of an internet-linked handheld computer critical care knowledge access system. *Critical Care* 8(6), R414-421. https://doi.org/10.1186/cc2967

[5] Naumanen, P., Savolainen H., Liesivuori J., 2008. Occupational Risk Identification Using Hand-Held or Laptop Computers. *International Journal of Occupational Safety and Ergonomics (JOSE)* 14:2, 207-215. https://doi.org/10.1080/10803548.2008.11076764

[6] Muslimin M.S., Nordin N.M., Mansor A.Z., Yunus M.M., 2017. The Design and Development of MobiEko: A Mobile Educational App for Microeconomics Module. *Malaysian Journal of Learning and Instruction*, 221–255. https://doi.org/10.32890/mjli.2017.7804

[7] ILO, 2015. Safety and Health at Work Mobile Apps. https://www.ilo.org/global/publications/apps/ [accessed 16 February 2023].

[8] Naiskodukaitse, 2018. Be prepared! Mobile application. https://www.naiskodukaitse.ee/Be_prepared_4292 [accessed 16 February 2023].

[9] Sanchis, R., García-Perales, Ó., Fraile, F., Poler, R., 2019. Low-Code as Enabler of Digital Transformation in Manufacturing Industry. *Applied Sciences* 10(1), 12. https://doi.org/10.3390/app10010012

[10] Vikebø, E., Sydvold, L. B., 2019. *An Inquiry into Low-Code Solutions in Institutions for Higher Education: a case study of low-code implementation at the Admissions Office at the Norwegian School of Economics.* Master Thesis, Norwegian School of Economics, Bergen, Norway. https://openaccess.nhh.no/nhh-xmlui/bitstream/handle/11250/2644682/masterthesis.PDF [Accessed 11 October 2022].

[11] Woo, M., 2020. The Rise of No/Low Code Software Development—No Experience Needed? *Engineering* 6(9), 960-961. https://doi: 10.1016/j.eng.2020.07.007

[12] Oltrogge, M., Derr, E., Stransky, C., Acar, Y., Fahl, S., Rossow, C., Pellegrino, P., Bugiel, S., Backes, M., 2018. The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators. In *2018 IEEE Symposium on Security and Privacy*, San Francisco, CA, USA. pp. 634-647. https://doi: 10.1109/SP.2018.00005

[13] Yajing, L., Peng, L., Chong, W., Mojtaba, S., Jing, Z., 2021. Characteristics and Challenges of Low-Code Development: The Practitioners' Perspective. In *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '21) October 11-15,* 2021, Bari, Italy. Article No.: 12, pp. 1–11 https://doi.org/10.1145/3475716.3475782

[14] *Glide. No Code App Builder*. https://www.glideapps.com/; 2019 [accessed 7.11.23].

[15] *Glide Community.* https://community.glideapps.com/; 2021 [accessed 7.11.23].

[16] *W3C. Badging API Unofficial Draft.* https://w3c.github.io/badging/; 2023 [accessed 7.11.23].

[17] *Web Periodic Background Synchronization.* https://wicg.github.io/background-sync/spec/PeriodicBackgroundSync-index.html ;2020 [accessed 7.11.23].

[18] *A guide to GDPR data privacy requirements in European Union*, 2023. https://gdpr.eu/tag/gdpr/ [accessed 7.11.23].

[19] *Top 10 Web Application Security Risks*, 2021. https://owasp.org/www-project-top-ten/ [accessed 7.11.23].

[20] Zerouali, A., Mens, T., Decan, A., De Roover, C., 2022. On the Impact of Security Vulnerabilities in the npm and RubyGems Dependency Networks. *Empirical Software Engineering*. 27(5):107. https://doi.org/10.1007/s10664-022-10154-1

[21] Jiyeon, L., Hayeon, K., Junghwan, P., Shin, I., Sooel, S., 2018. Pride and Prejudice in Progressive Web Apps: Abusing Native App-like Features in Web Applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Toronto, ON, Canada, pp. 1731-1746. https://doi.org/10.1145/3243734.3243867