Client-Side Gamification Engine for Enhanced Programming Learning

Ricardo Queirós
School of Media Arts and Design & CRACS – INESC TEC, Polytechnic of Porto, Portugal

Robertas Damaševičius
Department of Applied Informatics, Vytautas Magnus University, Vilnius, Lithuania

Rytis Maskeliūnas
Centre of Real Time Computer Systems, Kaunas University of Technology, Lithuania

Jakub Swacha
Department of IT in Management, University of Szczecin, Poland

Abstract

This study introduces the development of a client-based software layer within the FGPE project, aimed at enhancing the usability of the FGPE programming learning environment through client-side processing. The primary goal is to enable the evaluation of programming exercises and the application of gamification rules directly on the client-side, thereby facilitating offline functionality. This approach is particularly beneficial in regions with unreliable internet connectivity, as it allows continuous student interaction and feedback without the need for a constant server connection. The implementation promises to reduce server load significantly by shifting the evaluation workload to the client-side. This not only improves response times but also alleviates the burden on server resources, enhancing overall system efficiency. Two main strategies are explored: 1) caching the gamification service interface on the client-side, and 2) implementing a complete client-side gamification service that synchronizes with the server when online. Each approach is evaluated in terms of its impact on user experience, system performance, and potential security concerns. The findings suggest that while client-side processing offers considerable benefits in terms of scalability and user engagement, it also introduces challenges such as increased system complexity and potential data synchronization issues. The study concludes with recommendations for balancing these factors to optimize the design and implementation of client-based systems for educational environments.

2012 ACM Subject Classification Social and professional topics → Computer science education

Keywords and phrases Code generation, Computer Programming, Gamification

Digital Object Identifier 10.4230/OASIcs.ICPEC.2024.11

Funding This work is co-funded by the Erasmus+ Programme of the European Union within the project FGPEPlusPlus, with Agreement Number 2023-1-PL01-KA220-HED-000164696.

1 Introduction

Interactive and accessible learning platforms have become crucial in providing quality education to a geographically and economically diverse student population. Traditionally, such platforms rely heavily on constant server connectivity to function effectively, offering real-time feedback and interactive experiences that are central to modern pedagogical methodologies. However, this dependence on server-side processing presents significant challenges, particularly in areas with unstable internet access or limited server resources [32]. The evolution of client-side technologies, such as service workers and local storage, has introduced the potential for mitigating these issues by shifting some of the computational responsibilities from the server to the client. This shift not only promises to enhance the resilience of educational platforms against connectivity fluctuations but also aims to distribute the computational load more evenly, thus improving responsiveness and scalability [12]. The

advent of such technologies beckons a pivotal shift in the architecture of educational platforms, from a predominantly server-reliant model to a more decentralized approach [27, 7]. This transformation could revolutionize how educational services are delivered, ensuring more reliable and accessible learning experiences regardless of external network conditions.

The Framework for Gamified Programming Education (FGPE)[1] programming learning environment, thanks to the development of the FGPE Plus project[2] [21, 20], is currently provided as a Progressive Web Application, which means, once loaded, it could be used without sustained Internet connection. The problem is, for the students' solution of an exercise to be evaluated, and the relevant gamification rules triggered, there must be an active Internet connection at the time of submission. The goal of FGPE++[3] is to remove this barrier by providing a way to assess the programming exercise solutions client-side. This will both greatly improve the FGPE platform usability in areas in which users experience Internet connection problems and will also reduce the load on university servers, allowing to limit their role to batch processing of the student progress data on synchronization, instead of running the full evaluation server-side, which with many concurrent students on university servers having limited resources caused noticeable delays and sometimes even stalls that ruined all user experience. Consequently, this paves the way for providing large-scale gamified programming courses with limited technical resources, and will support the sustained growth of the number of the FGPE ecosystem users. The envisaged client-side software layer for the assessment of gamified programming exercises and gamification rule processing will be capable of producing instant feedback to the users even without active connection with the server, caching data on user progress and achievements, and synchronizing students' progress cached on the client with the global state stored at the server.

This study seeks to explore this transition within the context of the FGPE platform, aiming to develop a client-based version that maintains high functionality offline while synchronizing with the server when connectivity permits. This approach is particularly pertinent for educational institutions with limited IT infrastructure, as it allows them to offer a high-quality, interactive programming education with reduced dependence on robust internet services.

The rest of the article is structured in four sections: the second section presents the base project of all this work. The next section, depicts the most used strategies for supporting offline features in client-server applications. The following two sections present the execution plan for the development of the client-side evaluation engine, identifying two approaches. Finally, the contributions of this article to the scientific community are presented as well as the future work.

2 The FGPE environment

Framework for Gamified Programming Education (FGPE) is an open, programming-language-agnostic set of exercise formats, exemplary exercises, and the supporting software [29]. The framework addresses the needs of both students (for a more engaging programming education) and teachers (for a customizable web platform for gamified teaching of programming). The framework has been developed by a consortium of five European higher-education institutions: University of Szczecin (Szczecin, Poland), INESC TEC (Porto, Portugal), Aalborg University

[1] https://fgpe.usz.edu.pl/
[2] https://fgpeplus.usz.edu.pl/
[3] https://fgpeplus2.usz.edu.pl/

(Copenhagen, Denmark), University of Naples Parthenope (Naples, Italy), and Kaunas University of Technology (Kaunas, Lithuania), and financially supported from the Erasmus+ program[4].

The key software components of the FGPE ecosystem include:

- the editor (FGPE AuthorKit) which gives the exercise creators the capability to design and implement both exercises and gamified scenarios with an intuitive wizard-like form-based user interface [25],
- gamification service (FGPE GS) that processes gamification rules and manages the overall game state [24],
- the interactive learning environment (FGPE PLE), which lets students access gamified exercises, solve them, and receive graded feedback, and lets teachers organize exercise sets, grant students access, and monitor their learning progress [21].

Figure 1 provides a comprehensive visualization of the FGPE architecture.

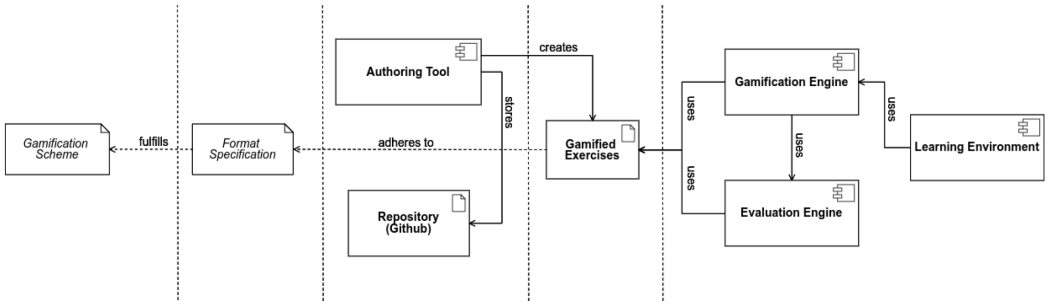

Figure 1 FGPE Architecture.

Recently, FGPE platform has been enriched with an LTI-compliant interface allowing the FGPE-run exercises to be embedded in LMS-based courses, developed using any of the popular systems (e.g., Moodle or Open edX) [17].

To evaluate the solutions of the programming exercises submitted by students, the FGPE platform uses the reliable and secure Mooshak system [16]. As both the FGPE GS and Mooshak are run server-side, although the students can edit their code in FGPE PLE, which is currently provided as Progressive Web App, without the Internet access, they cannot submit it and get feedback until the connection is restored, which can be an issue for users traveling or staying in areas with poor Internet connectivity.

This defines the problem that the research presented in this paper aims to address.

3 Related work

This section reviews the existing literature and developments in the area of offline capabilities for educational platforms and the broader spectrum of tools that facilitate offline functionality in client-server applications. It sets the groundwork for understanding the technological and conceptual frameworks that our research builds upon.

[4] https://fgpeplus2.usz.edu.pl/

3.1 Offline Strategies

The evolution from traditional client-server setups to architectures where significant functionalities are executed on the client-side has garnered considerable attention in research circles [15]. This paradigm shift holds particular significance in educational technology landscapes, where the necessity for continuous server interaction can pose accessibility and usability challenges, especially in areas with limited connectivity.

One pivotal approach within this realm revolves around leveraging service workers, which are scripts operating in the background on the client side, independent of the active web page [10]. These workers facilitate resource caching and streamline data synchronization processes, thus augmenting user experiences by enabling interactive engagements even in offline scenarios. Subsequently, they efficiently synchronize data with the server upon restoration of connectivity [23]. In addition to service workers, the adoption of local storage mechanisms and indexed databases such as IndexedDB stands out as another prominent strategy [2]. These technologies empower educational platforms to locally store substantial volumes of data, ranging from user progress metrics to interactive state information. By doing so, they alleviate the reliance on continuous data retrieval from the server, thereby enhancing system efficiency and responsiveness [13]. Finally, the integration of differential synchronization emerged as an interesting concept in this landscape [3]. This approach entails transmitting only the changes or differences between the client and server, rather than entire datasets. By minimizing data transfer requirements, it significantly contributes to optimizing system performance and bolstering application responsiveness [33].

3.2 Other Tools

In addition to strategies tailored explicitly for offline functionality, a plethora of tools and frameworks exist to support the development of client-heavy applications. Google's Workbox stands out among these, offering libraries and Node modules that streamline the management of service workers, making it more accessible and robust. Workbox boasts extensive features for precaching, runtime caching, and efficient data retrieval and storage strategies [1]. React, a JavaScript library renowned for building user interfaces, plays a crucial role in creating responsive and dynamic client-side applications. Its capability to manage state locally and render components based on user interactions makes it a preferred choice for developing educational platforms with minimal reliance on server-side rendering [31]. Frameworks such as Electron broaden the horizons by enabling the creation of native applications using web technologies. These applications can function as standalone desktop applications, eliminating the need for an active internet connection. By harnessing Electron, web-based educational platforms can seamlessly transition into fully functional desktop applications [28].

Collectively, these tools and strategies align with the overarching objective of bolstering the resilience and accessibility of educational platforms [11], especially in environments grappling with limited connectivity. Our study endeavors to leverage these advancements to propose a novel approach to managing offline functionalities within the FGPE platform, with the aim of delivering a seamless and uninterrupted learning experience.

4 The offline client-server optimization model

Bellow we explain optimization problem with five objective functions to maximize, representing usability, performance, server load, autonomy, and security. It defines decision variables and constraints related to two options, each with specific criteria and thresholds.

Through our **objective functions**, we aim to maximize the following criteria:

1. Usability (C_1)
2. Performance (C_2)
3. Server Load (C_3)
4. Autonomy (C_4)
5. Security (C_5)

Let us define **decision variables** used in our model:

- x_1 represent Option 1: Cached interface for Gamification Service
- x_2 represent Option 2: Client-side Gamification Service

Further we will define **constraints** for each option x_i, where $i = 1, 2$, related to:

- Client-side Complexity (D_1, D_6)
- Data Synchronization (D_2, D_7)
- Control (D_3, D_8)
- Duplication of Logic (D_4, D_9)
- Resource Requirements (D_5, D_{10})

We have chosen Non-dominated Sorting Genetic Algorithm II (NSGA-II [5] to explore the solution space by maintaining a diverse population of non-dominated solutions and iteratively evolving it towards the Pareto optimal front. It is well-suited for multi-objective optimization problems like the FGPE approach, where multiple conflicting objectives need to be considered simultaneously.

The **objective function** for NSGA-II combines the weighted sum of criteria:

$$\begin{aligned}\text{Maximize } f_1(x) = {} & w_1 \cdot C_1(x) + w_2 \cdot C_2(x) + w_3 \cdot C_3(x) \\ & + w_4 \cdot C_4(x) + w_5 \cdot C_5(x)\end{aligned}$$

where $C_i(x)$ represents the value of criterion i for option x, and w_i are the weights assigned to each criterion.

The **constraints** ensure that each option satisfies the respective thresholds: D_1, D_2, D_3, D_4, D_5 for x_1, and $D_6, D_7, D_8, D_9, D_{10}$ for x_2.

Below is a pseudo-code representation of a multi-objective evolutionary algorithm inspired by NSGA-II for solving the optimization problem described:

Algorithm 1 Multi-Objective Evolutionary Algorithm.

```
Initialize population P randomly or using a heuristic method
Evaluate(P) // Evaluate the objective functions for each individual considering the constraints
repeat
    Create an empty offspring population Q
    while |Q| < |P| do
        Select parents from P based on non-dominated sorting and crowding distance, considering the constraints
        Perform crossover and mutation to create offspring, ensuring constraints are satisfied
        Evaluate(Q) // Evaluate the objective functions for each offspring considering the constraints
        Merge P and Q to form R (combined population)
        Perform non-dominated sorting on R considering the constraints
        Select top |P| individuals from R based on non-domination and crowding distance
    end while
    Replace P with the selected individuals from R
until termination criteria are met
```

In the pseudo-code:

- *Initialize* function initializes the population of candidate solutions considering the decision variables.
- *Evaluate* function evaluates the objective function values for each individual in the population, taking into account the constraints.
- *Select* function selects parents for reproduction based on non-dominated sorting and crowding distance, ensuring feasibility with constraints.
- *Crossover* and *Mutation* functions create offspring from selected parents while satisfying the constraints.
- *Merge* function combines the parent population P and offspring population Q to form a combined population R.
- *Non-dominated sorting* sorts individuals in R based on non-domination, creating fronts of non-dominated solutions, considering the constraints.
- *Replace* function selects top individuals from R to form the next generation population P, based on non-domination and crowding distance, while ensuring feasibility with constraints.
- The algorithm continues iterating until a termination condition is met, such as reaching a maximum number of generations or evaluations.

5 The implementation of Client-side Evaluation Engine

In the scope of the FGPE project, two options were identified:

- Client-side execution environment + cached interface for Gamification Service
- Client-side execution environment + client-side Gamification Service (synchronized with server when available)

Both are detailed in the next subsections.

5.1 Cached interface for Gamification Service

This option focuses on caching the interface for the Gamification Service on the client-side. It means that even if there's no active internet connection, the user can still interact with the interface and work on their exercises. The client-side execution environment would allow the user to submit their solution regardless of the server's availability. The cached interface would ensure that the user can still receive feedback and engage with gamification elements locally, enhancing usability in areas with poor internet connectivity.

Bellow we explore the key elements behind our approach (see Figure 2 for reference).

1. First, the system must have offline functionality. The client-side application needs to store the interface components, assets, and relevant data locally using technologies like IndexedDB, Web Storage, or Service Workers. Progressive Web Apps (PWAs) [19] are employed to ensure key interface elements are cached for offline use. Service Workers are used rto intercept network requests, allowing the application to function even without an active internet connection by serving cached resources.
2. Next we must ensure data synchronization by implementing robust synchronization mechanisms [8] to ensure data consistency between the client and server once the internet connection is restored. We have used differential synchronization to sync data while minimizing bandwidth usage and conflicts. Conflict resolution strategies were employed to handle scenarios where changes are made both locally and on the server during offline usage.

3. Third element was solving client-side complexity as caching the interface on the client-side introduces additional complexity to the frontend codebase [18]. We have used modularization to help manage this complexity by breaking down the application into smaller, manageable parts.
4. Fourth element was security considerations, as with increased client-side execution, security risks such as data tampering, injection attacks, and unauthorized access become more pronounced [30]. We have implemented secure communication protocols, data encryption, and input validation on both client and server sides to mitigate these risks.
5. Fifth element was server load reduction, which we solved by offloading interface rendering and user interaction to the client-side [22], thus the server's role shifts to primarily handling data processing and authentication. This led to reduced server load and improved scalability, as the server no longer needed to handle as many concurrent interface requests, leaving our server to efficiently handle batch processing tasks and scale with increasing user demand.
6. Final element is usability and performance. Main improvement was done by caching the interface locally, which enhances usability in areas with poor internet connectivity, providing users with a seamless experience [26]. Performance gains were also achieved by reducing latency associated with fetching interface resources from the server, especially for frequently accessed components using cache policies.

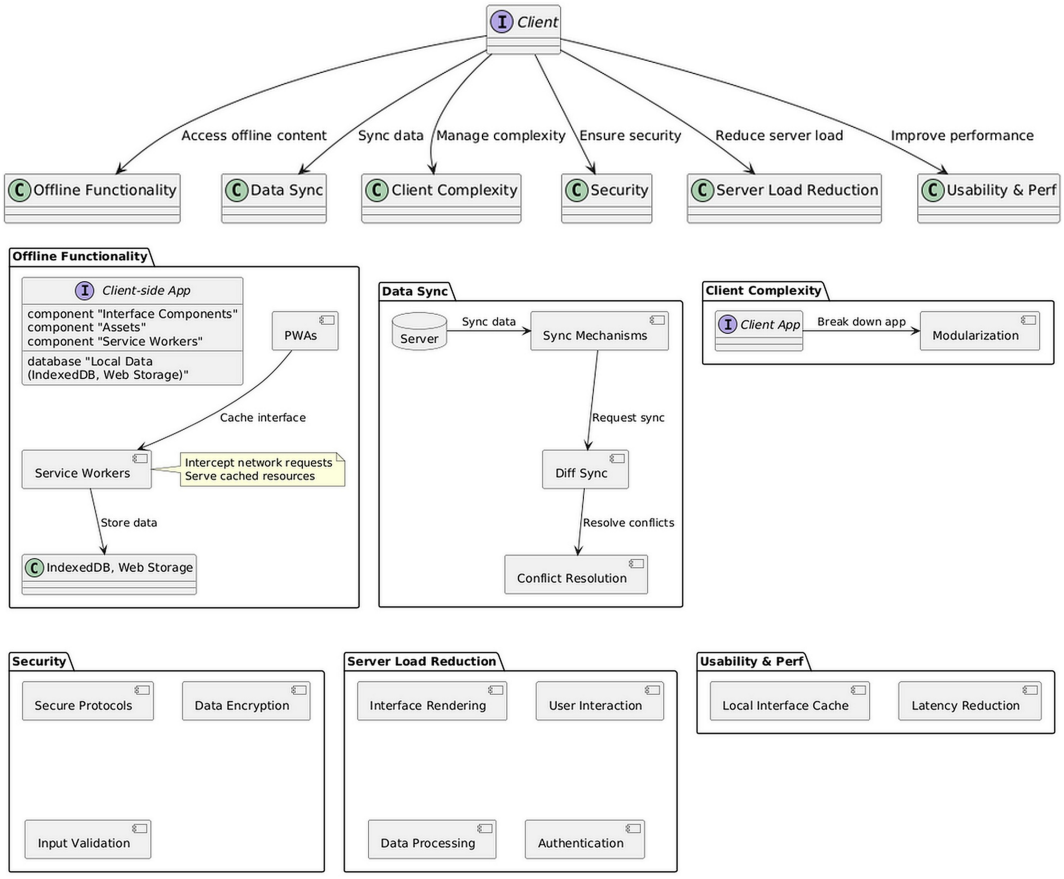

Figure 2 Cached interface for Gamification Service.

Following this approach can lead to two main advantages:

- Seamless user experience: users can continue working on exercises without interruptions due to server downtime or poor internet connection.
- Reduced server load: by caching the interface locally, the server's role is minimized to batch processing, reducing the strain on resources and potentially improving overall performance.

However, there are also disadvantages such as increased client-side complexity, data synchronization challenges and limited server-side control. Firstly, implementing a gamification service cached interface on the client-side adds complexity to the frontend codebase, potentially making maintenance and debugging more challenging. Secondly, caching the interface for the Gamification Service locally may introduce challenges in synchronizing data between the client and server when an internet connection is available. Finally, when relying on client-side execution, there may be limited control from the server-side, which could potentially lead to security vulnerabilities in data management.

5.2 Client-side Gamification Service

This option involves implementing the Gamification Service entirely on the client-side, synchronized with the server when an internet connection is available. It allows for more autonomy on the client-side, as the gamification elements are not dependent on server availability. However, synchronization would be necessary to update the global state and ensure consistency across users (see Figure 3 for reference).

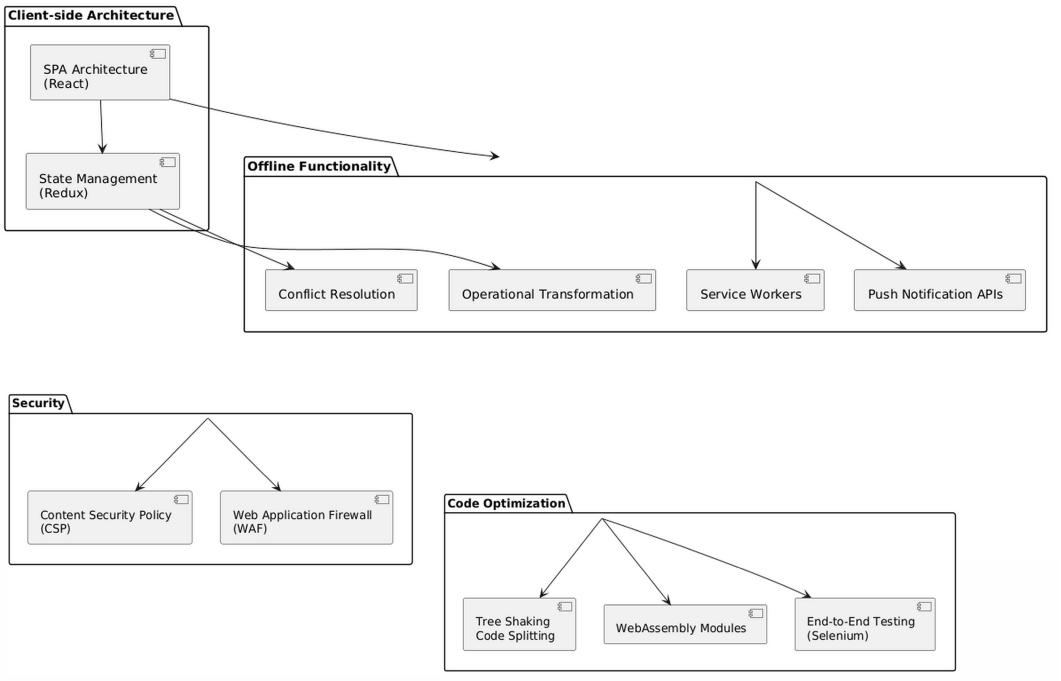

Figure 3 Client-side Gamification Service.

To implement efficient Gamification Service entirely on the client-side we needed an advanced Client-side Architecture. We have chosen single page application (SPA) architecture [14], based on React for efficient client-side rendering and seamless user experience. State management was implemented using Redux to manage complex gamification state across different components of the application.

Offline Functionality was first assured with conflict resolution mechanisms [4] to handle data conflicts that may arise during offline usage, ensuring data integrity when synchronizing with the server. We have also implemented operational transformation algorithms to reconcile concurrent edits made to the gamification data by multiple clients during offline usage, minimizing data inconsistencies. The progressive web application (PWA) had service workers implemented to enable offline caching and background synchronization of gamification data, transforming the application into a full-fledged PWA. We have also considered integrating push notification APIs to notify users of gamification updates and achievements, enhancing user engagement and retention.

Security was assured by applying content security policy (CSP) [6] to mitigate cross-site scripting (XSS) attacks and prevent execution of unauthorized scripts, enhancing the security of the client-side code. Web Application Firewall (WAF) was also used to monitor and filter HTTP traffic, detecting and blocking malicious requests targeting the client-side gamification service.

Code optimization was done through tree shaking and code splitting to eliminate dead code and reduce bundle size, optimizing performance and loading times of the client-side application. We have also integrated WebAssembly modules for computationally intensive gamification algorithms, leveraging the performance benefits of low-level bytecode execution in modern web browsers. End-to-end testing was done using Selenium to validate the functionality and performance of the client-side gamification service across different scenarios and environments.

Using this approach has the following advantages:

- Increased autonomy: Users can receive instant feedback and engage with gamification elements even without an internet connection, as the service is entirely client-side.
- Reduced dependency on server: By moving the gamification logic to the client-side, there's less reliance on server availability, potentially reducing downtime and improving user experience.

This approach lead also to several disadvantages such as security risks, duplication of logic and dependency on client resources. In the former, handling sensitive gamification logic and data on the client-side could pose security risks, as client-side code is inherently less secure and more susceptible to tampering or exploitation compared to server-side code. Other pitfall is the duplication of logic, since moving the gamification logic entirely to the client-side may result in duplication of code and logic, as similar functionalities may need to be implemented both on the client and server sides. This lead to code redundancy and maintenance overhead. Finally, relocating the gamification service to the client-side may increase the resource requirements on the client devices, especially for complex gamification algorithms or large datasets. This could impact performance on devices with limited processing power or memory.

6 Conclusions

The main result of this work is sketching the blueprint for the development of a client-side software layer for the assessment of gamified programming exercises within the FGPE project. This component will be responsible for producing instant feedback to the users even without active connection with the server, caching data on user progress and achievements, and synchronizing students' progress cached on the client with the global state stored at the server.

In order to tackle this challenge, two options were identified: 1) caching the interface for the Gamification Service on the client-side and 2) implementing the Gamification Service entirely on the client-side.

While both options offer solutions to enable client-side execution and offline capabilities, they also come with their own set of challenges and drawbacks. The choice between the two options should consider factors such as the specific requirements of the application, the desired level of control and security, and the technical capabilities (and constraints) of the client devices and server infrastructure.

The software is planned was developed using two-way communication between web browser and web server [9]. The software will be freely distributed on the GitHub platform in the FGPE repository[5] under the GNU General Public License v3 open-source license[6], so that any educational institution can use it and, if necessary, adopt it for their specific purposes free of charge.

References

1 Workbox: Powerful tools for your service workers, 2021.

2 Thamer Al-Rousan. An investigation of user privacy and data protection on user-side storage, 2019.

3 Wahab Kh Arabo. The web and ai influences on distributed consensus protocols in cloud computing: A review of challenges and opportunities. *Journal of Information Technology and Informatics*, 3(1), 2024.

4 Rebecca Kai Cassar, Joseph Vella, and Joshua Ellul. A conflict resolution abstraction layer for eventually consistent databases. In *2016 International Conference on Engineering & MIS (ICEMIS)*, pages 1–5. IEEE, 2016.

5 Sankhadeep Chatterjee, Sarbartha Sarkar, Nilanjan Dey, Amira S Ashour, and Soumya Sen. Hybrid non-dominated sorting genetic algorithm: Ii-neural network approach. In *Advancements in Applied Metaheuristic Computing*, pages 264–286. IGI Global, 2018.

6 Hsing-Chung Chen, Aristophane Nshimiyimana, Cahya Damarjati, and Pi-Hsien Chang. Detection and prevention of cross-site scripting attack with combined approaches. In *2021 International Conference on Electronics, Information, and Communication (ICEIC)*, pages 1–4. IEEE, 2021.

7 Xiaodan Chen, Jun Lu, Mei Gong, Benjun Guo, and Yuanping Xu. Design and implementation of decentralized online education platform. In *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, 2020. doi:10.1109/ICMCCE51767.2020.00212.

8 Mohammad Faiz and Udai Shanker. Data synchronization in distributed client-server applications. In *2016 IEEE International Conference on Engineering and Technology (ICETECH)*, pages 611–616. IEEE, 2016.

9 Ian Fette and Alexey Melnikov. The websocket protocol. Technical Report RFC 6455, IETF, 2011.

10 Joy M Field, Liana Victorino, Ryan W Buell, Michael J Dixon, Susan Meyer Goldstein, Larry J Menor, Madeleine E Pullman, Aleda V Roth, Enrico Secchi, and Jie J Zhang. Service operations: what's next? *Journal of Service Management*, 29(1):55–97, 2018.

11 Miftachul Huda. Between accessibility and adaptability of digital platform: investigating learners' perspectives on digital learning infrastructure. *Higher Education, Skills and Work-Based Learning*, 14(1):1–21, 2024.

12 Mayssa Jemel and A. Serhrouchni. Toward user's devices collaboration to distribute securely the client side storage. In *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*, 2015. doi:10.1109/NOTERE.2015.7293479.

[5] https://github.com/FGPE-Erasmus

[6] https://www.gnu.org/licenses/gpl-3.0.en.html

13 Young joo Shin and Kwangjo Kim. Differentially private client-side data deduplication protocol for cloud storage services. *Secur. Commun. Networks*, 8:2114–2123, 2015. doi:10.1002/sec.1159.

14 DV Kornienko, SV Mishina, and MO Melnikov. The single page application architecture when developing secure web services. In *Journal of Physics: Conference Series*, volume 2091(1), page 012065. IOP Publishing, 2021.

15 Raoni Kulesza, Marcelo Fernandes de Sousa, Matheus Lima Moura de Araújo, Claudiomar Pereira de Araújo, and Aguinaldo Macedo Filho. Evolution of web systems architectures: a roadmap. *Special Topics in Multimedia, IoT and Web Technologies*, pages 3–21, 2020.

16 José Paulo Leal and Fernando Silva. Mooshak: A web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003.

17 José Paulo Leal, Ricardo Queirós, Pedro Ferreirinha, and Jakub Swacha. A Roadmap to Convert Educational Web Applications into LTI Tools. In *Third International Computer Programming Education Conference*, ICPEC 2022, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASICS.ICPEC.2022.12.

18 Vittorio Maniezzo, Marco A Boschetti, Antonella Carbonaro, Moreno Marzolla, and Francesco Strappaveccia. Client-side computational optimization. *ACM Transactions on Mathematical Software (TOMS)*, 45(2):1–16, 2019.

19 Rytis Maskeliunas, Robertas Damasevicius, Tomas Blazauskas, and Jakub Swacha. Evaluation of a progressive web application for gamified programming learning. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*, pages 1334–1334, 2022.

20 Rytis Maskeliunas, Robertas Damasevicius, Tomas Blazauskas, and Jakub Swacha. Evaluation of a progressive web application for gamified programming learning. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education, Volume 2, SIGCSE 2023, Toronto, ON, Canada, March 15-18, 2023*, volume 2, page 1334, 2023. doi:10.1145/3545947.3576385.

21 Rytis Maskeliūnas, Robertas Damaševičius, Tomas Blažauskas, Jakub Swacha, Ricardo Queirós, and José Carlos Paiva. FGPE+: The mobile FGPE environment and the pareto-optimized gamified programming exercise selection model—an empirical evaluation. *Computers*, 12(7), 2023. doi:10.3390/computers12070144.

22 Farouk Messaoudi, Adlen Ksentini, and Philippe Bertin. Toward a mobile gaming based-computation offloading. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018.

23 J. Moscicki and L. Mascetti. Cloud storage services for file synchronization and sharing in science, education and research. *Future Gener. Comput. Syst.*, 78:1052–1054, 2018. doi:10.1016/j.future.2017.09.019.

24 José Carlos Paiva, Alicja Haraszczuk, Ricardo Queirós, José Paulo Leal, Jakub Swacha, and Sokol Kosta. FGPE Gamification Service: A graphql service to gamify online education. In *Trends and Applications in Information Systems and Technologies: Volume 4*, pages 480–489, Cham, Switzerland, 2021. Springer.

25 José Carlos Paiva, Ricardo Queirós, José Paulo Leal, Jakub Swacha, and Filip Miernik. Managing gamified programming courses with the FGPE platform. *Information*, 13(2):45, 2022.

26 Adrian Petcu, Madalin Frunzete, and Dan Alexandru Stoichescu. Evolution of applications: From natively installed to web and decentralized. In *International Conference on Computational Science and Its Applications*, pages 253–270. Springer, 2023.

27 U. Rahardja, M. A. Ngadi, R. Budiarto, Q. Aini, Marviola Hardini, and Fitra Putri Oganda. Education exchange storage protocol: Transformation into decentralized learning platform. *Frontiers in Education*, 6, 2021. doi:10.3389/feduc.2021.782969.

28 T. Sproull and Bill Siever. Going native with your web dev skills: An introduction to react native for mobile app development, 2020.

29 Jakub Swacha, Thomas Naprawski, Ricardo Queirós, José Carlos Paiva, José Paulo Leal, Ciro Giuseppe De Vita, Gennaro Mellone, Raffaele Montella, Davor Ljubenkov, and Sokol Kosta. Open Source Collection of Gamified Programming Exercises. In *Proceedings of the thirty-seventh Information Systems Education Conference*, ISECON 2021, pages 120–123, Chicago, IL, USA, 2021. Foundation for IT education.

30 Hamed Tabrizchi and Marjan Kuchaki Rafsanjani. A survey on security challenges in cloud computing: issues, threats, and solutions. *The journal of supercomputing*, 76(12):9493–9532, 2020.

31 Mohit Thakkar. Reactjs: A comprehensive analysis of its features, performance, and suitability for modern web development, 2020.

32 B. Wang and Yimin Zhou. Research and implementation of multiple virtual management platform in multimedia classroom based on cloud storage. In *2011 International Conference on Multimedia Technology*, 2011. doi:10.1109/ICMT.2011.6002332.

33 Tian Wang, Jiyuan Zhou, Anfeng Liu, Md Zakirul Alam Bhuiyan, Guojun Wang, and W. Jia. Fog-based computing and storage offloading for data synchronization in iot. *IEEE Internet of Things Journal*, 6:4272–4282, 2019. doi:10.1109/JIOT.2018.2875915.