Enhanced Resource Management for Web Based Thin Clients Using Cross-Platform Progressive Offline Capabilities

George Alex Stelea[1(✉)], Maurizio Murroni[2], Vlad Popescu[1], Titus Balan[1], and Vlad Fernoaga[1]

[1] Transilvania University, bd Eroilor 29A, Brasov, Romania
george.stelea@unitbv.ro
[2] University of Cagliari, Via San Giorgio 12/2, 09124 Cagliari, Italy

Abstract. Web based thin clients are applications delivering content from the Internet or Intranet and accessed via the browser on the running end device. These clients are portable and cross-device compatible and have a large spectrum of applications, can perform from tele-measurement tasks to management and information centralization. The capability of web-based thin clients to function offline is a requirement that is indispensable even today for many companies because offline-enabled thin clients allow the users to continue working without workflow disturbance, preventing the loss of data, even when the connection to the Internet is missing or malfunctioning. This paper is dedicated to a "barrier-free" cross-platform responsive and progressive web based thin client, presenting its architecture and development, as well as the offline capabilities using caching techniques and its advantages in resource management and information back-up and security.

Keywords: Web based thin client · Cross-platform progressive offline capabilities · Enhanced resource management

1 Introduction

Thin clients are necessary when fat clients are too expensive and upscale or because they need more computing power or energy than available from the low-end terminals (e.g. tablets or smartphones). A web based thin client is an application program functioning according to the client-server model [1], where all the software is running on the server, and only the presentation is delivered on the device (e.g. GIS or tele-measurement applications where the thin client is basically the web browser) [2]. In the modern Web, oriented towards portables, Cloud computing and virtualization the trend is towards thin clients. Unlike traditional desktop software, web based thin clients do not need important installation and execution processes on the user's machine [3]. Instead, the data processing and evaluation mainly takes place on a remote web server [4] and only the result of the data processing is transmitted to the user's local client computer for display or output [5], usually via a web browser which handles the

A. Kliks et al. (Eds.): CrownCom 2019, LNICST 291, pp. 361–372, 2019.
https://doi.org/10.1007/978-3-030-25748-4_27

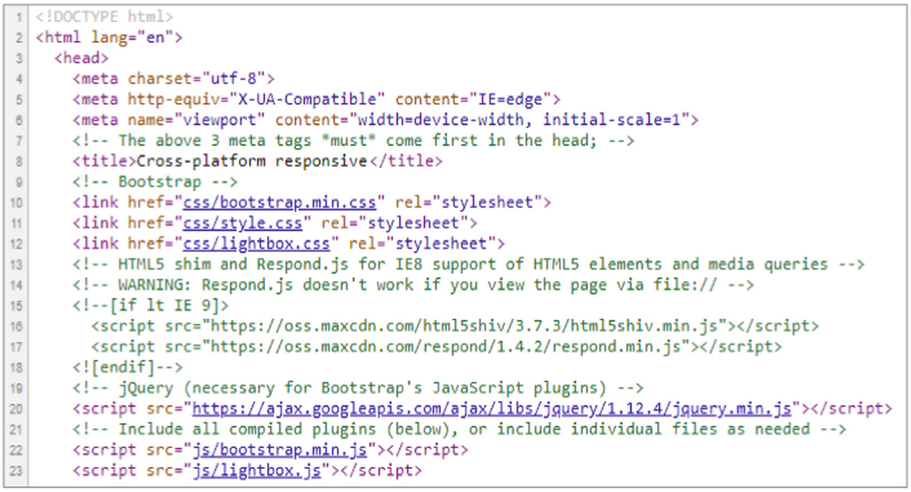
communication with the web server (via the HTTP protocol) as well as the representation of the user interface [6].

On the server or virtualized desktop, the inputs are processed and the output is sent back to the client, who only has to display them. The current generation of terminal servers or virtualization solutions also allows the use of hardware beyond a printer and works with optimized methods for playback of audio or video data. Although the trend in the age of IoT networks and mobile/smart devices today is that everyone is online 24 h a day [7], there are still many situations in which an Internet connection is malfunctioning or temporary missing (e.g. on the go, in cellular radio communications). An offline-enabled thin client allows the user to maintain the session – continuing to work even in this case without being disturbed in his workflow and without the loss of data.

In this paper, we present a "barrier-free" [8] cross-platform responsive and progressive web based thin client using the new JavaScript [9] and HTML5 [10] specifications with the Online Application Caching API, "Service Workers" and Bootstrap Framework, presenting its architecture, development environment and its advantages in resource management, rich context information administration and enhanced security. The paper is structured as follows: Sect. 2 presents the concept and the technology to achieve the cross-platform content to terminal adaption and the aim of this practice, Sect. 3 outlines the progressive web methodology, describing the combined possibilities offered by most modern browsers with the benefits of mobile use, Sect. 4 details the offline capabilities of the proposed solution using a caching system, retrieving and intercepting network requests from the cache and delivering push messages, indicating the increased maintainability and testability, while Sect. 5 describes the conclusions and the future work to be done.

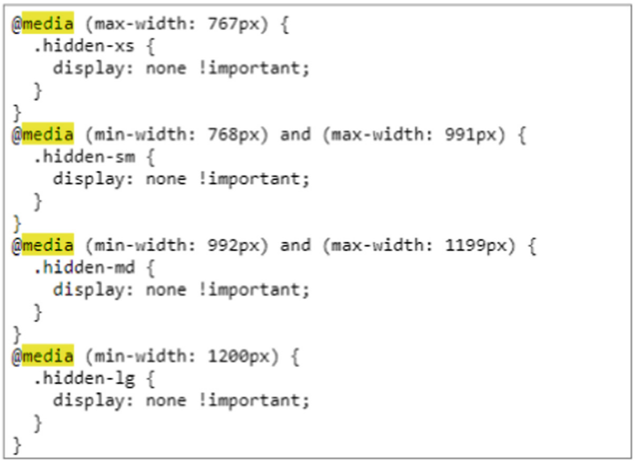
2 Content to Terminal Adaption

The size and resolution of displays on laptops, desktops, tablets, smartphones and TV sets can vary considerably [11]. For this reason, the appearance and operation of a web based thin client are very dependent on the different requirements of the devices [12]. The aim of the content to terminal adaption practice is that thin clients adapt their presentation so that they present themselves as clear and user-friendly as possible to each viewer [13]. The criteria for the customized appearance are, in addition to the size of the display device, also the available input methods (touch screen, mouse) or the bandwidth of the Internet connection [14]. In order to achieve the responsive and adaptive design of the solution we have chosen the Bootstrap Framework and CSS3 Media Queries - as shown in Figs. 1 and 2 - that allow different designs depending on certain characteristics of the output environment.



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- The above 3 meta tags *must* come first in the head; -->
    <title>Cross-platform responsive</title>
    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet">
    <link href="css/style.css" rel="stylesheet">
    <link href="css/lightbox.css" rel="stylesheet">
    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
    <!--[if lt IE 9]>
      <script src="https://oss.maxcdn.com/html5shiv/3.7.3/html5shiv.min.js"></script>
      <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
    <![endif]-->
    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
    <!-- Include all compiled plugins (below), or include individual files as needed -->
    <script src="js/bootstrap.min.js"></script>
    <script src="js/lightbox.js"></script>
```

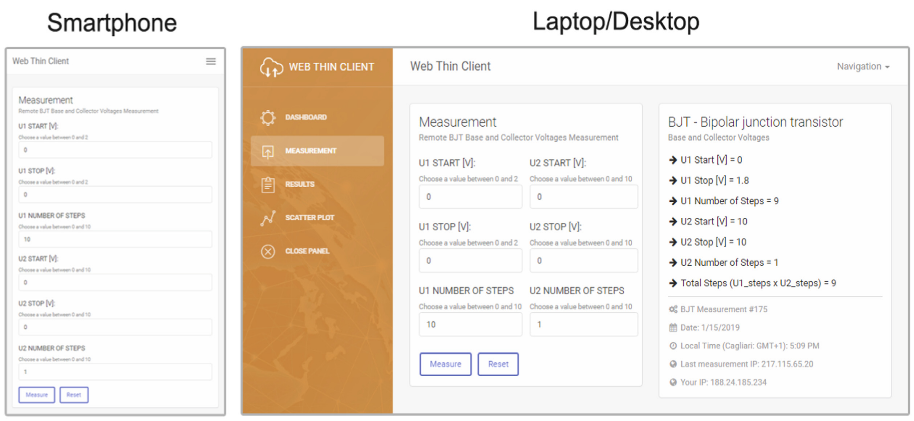
Fig. 1. Bootstrap Framework installation and call in the web applications <head> section.



```
@media (max-width: 767px) {
  .hidden-xs {
    display: none !important;
  }
}
@media (min-width: 768px) and (max-width: 991px) {
  .hidden-sm {
    display: none !important;
  }
}
@media (min-width: 992px) and (max-width: 1199px) {
  .hidden-md {
    display: none !important;
  }
}
@media (min-width: 1200px) {
  .hidden-lg {
    display: none !important;
  }
}
```

Fig. 2. CSS3 media queries.

In order to achieve optimal recognition using CSS for all display formats, the media information with media queries was requested before loading the application. It was not necessary to record the appropriate screen size for each individual device, but rather the types of devices, media features and breakpoints. The terminal then automatically loads the correct part of the CSS file and displays the content as the size of the screen allows. The thin client's user interface (UI) uses custom web services for graphical indicators, meters, text boxes, inputs and buttons, which are also adaptable to various devices, as shown in Fig. 3.



Web Thin Client

Measurement
Remote BJT Base and Collector Voltages Measurement

U1 START [V]:
Choose a value between 0 and 2
0

U1 STOP [V]:
Choose a value between 0 and 2
0

U1 NUMBER OF STEPS
Choose a value between 0 and 10
10

U2 START [V]:
Choose a value between 0 and 10
0

U2 STOP [V]:
Choose a value between 0 and 10
0

U2 NUMBER OF STEPS
Choose a value between 0 and 10
1

BJT - Bipolar junction transistor
Base and Collector Voltages

→ U1 Start [V] = 0
→ U1 Stop [V] = 1.8
→ U1 Number of Steps = 9
→ U2 Start [V] = 10
→ U2 Stop [V] = 10
→ U2 Number of Steps = 1
→ Total Steps (U1_steps x U2_steps) = 9

BJT Measurement #175
Date: 1/15/2019
Local Time (Cagliari: GMT+1): 5:09 PM
Last measurement IP: 217.115.65.20
Your IP: 188.24.185.234

Fig. 3. "Thin client" solution display of a tele-measurement on a smartphone and laptop/desktop device resolution.

This is how cross-platform content to terminal adaption was implemented, and, because there are used only standardized technologies, the software will also be compatible with subsequent devices that would be built using this standards.

3 Management Trough Progressive Web Methodology

Using this application model we aimed to combine the possibilities offered by most modern browsers with the benefits of mobile use [15]. The term "progressive" refers to the fact that, from the point of view of the user experience, the thin clients progressively adapts itself to the device and has many features, including speed and device optimization, that were previously reserved only for native software. We have combined, this way, the advantages of a classic thin client and of a custom desktop/mobile application. To implement the previously described feature, standardized HTML5, CSS3 and JavaScript was used, as shown in Fig. 4.

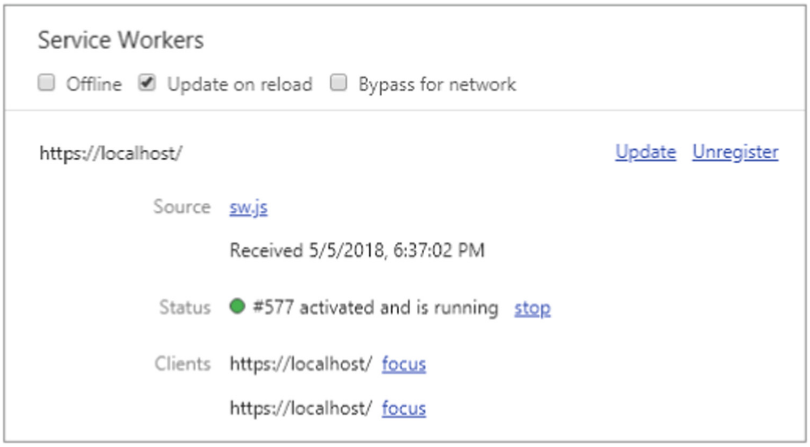
```
async function registerSW() {
  if ('serviceWorker' in navigator) {
    try {
      await navigator.serviceWorker.register('./sw.js');
    } catch (e) {
      alert('ServiceWorker registration failed.');
    }
  } else {
    document.querySelector('.alert').removeAttribute('hidden');
  }
}
```

Fig. 4. JavaScript asynchronous registration function.

In addition, we have used “Service Workers” [16] to serve through optimal caching of the online functionalities, as presented in Fig. 5. A service worker is a programmable network proxy, allowing the network requests control from the application. It is terminated when not in use, and restarted/executed on access and custom events. The HTTPS protocol was used for secure communication between the web client and the web server.

For example, the user starts the application in a browser, enters the URL of the web server and sends the first request. The web server accepts this request and passes it to the thin client, which initially acts like a web application. The thin client then generates or loads the HTML source code of the requested resources, which are sent back to the user’s browser by the web server (HTTPS response).

Due to the responsive design, the users see a software product layout adapted to its terminal. Although the web based thin client is accessed through an URL, the users can drag an icon to their mobile device home screen or receive push notifications and use the application offline. Progressive enhancement technology allows for each user the best possible user experience based on their technical capabilities.


Service Workers

☐ Offline ✓ Update on reload ☐ Bypass for network

https://localhost/ Update Unregister

Source sw.js

Received 5/5/2018, 6:37:02 PM

Status ● #577 activated and is running stop

Clients https://localhost/ focus

https://localhost/ focus

Fig. 5. Browser console - service worker verification.

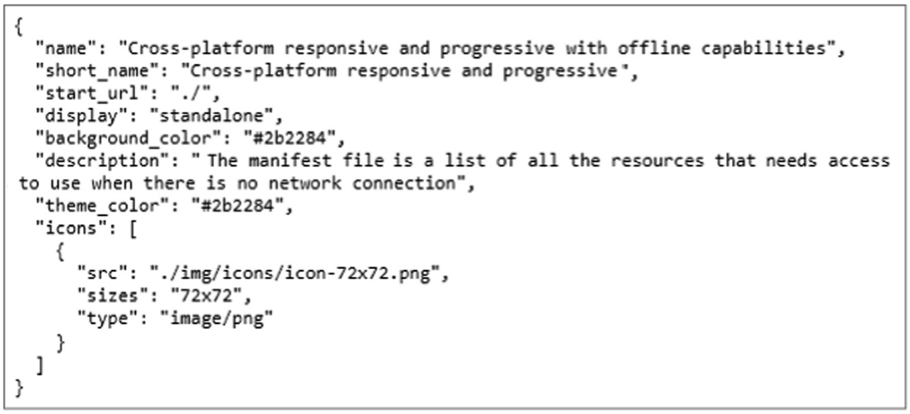
The main advantages of using progressive enhancement on the thin client are:

- Custom Management and Updates – data and information is easy to manage and the resources are always up-to-date thanks to the data update process offered by Service Workers;
- Decentralized backups - a user can work in a different workplace every day without any restrictions using local storage and local databases;
- Endpoint Management - is optimized because on the thin client is running only the software necessary for server communication;
- Progressive - it works for every user, regardless of the browser chosen because it is built at the base with progressive improvement principles;

– Responsive - it adapts to the various screen sizes: desktop, mobile, tablet, or even dimensions that can later become available;
– Independent of connection availability - Service Workers allow the application to run online, in intermittent connections (with longer interrupts) or with low quality connections.
– Secure - exposed over HTTPS protocol to prevent the connection from displaying unwanted information or altering the contents;
– Discoverable - it is identified as an "application" thanks to the W3C manifesto and the Service Worker registration scope that allows search engines to find them;
– Linkable - easily shared via the URL, not requiring complex installations.

4 Offline Capabilities

The engine of the offline capabilities is based on a cache manifest file as presented in Fig. 6. The manifest file is a list of all the resources that the thin client needs to access when there is no network connection [17]. This can be a common text file (or a JSON file) located elsewhere on the web server.



```
{
  "name": "Cross-platform responsive and progressive with offline capabilities",
  "short_name": "Cross-platform responsive and progressive",
  "start_url": "./",
  "display": "standalone",
  "background_color": "#2b2284",
  "description": " The manifest file is a list of all the resources that needs access
to use when there is no network connection",
  "theme_color": "#2b2284",
  "icons": [
    {
      "src": "./img/icons/icon-72x72.png",
      "sizes": "72x72",
      "type": "image/png"
    }
  ]
}
```

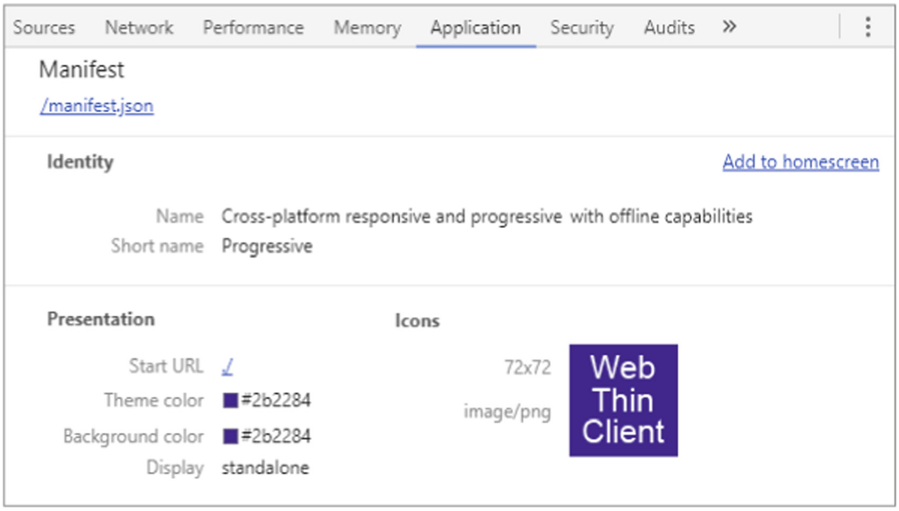
Fig. 6. JSON cache manifest file.

In online mode the thin client reads the list of URLs from the manifest file, downloads the resources, caches them locally, and automatically keeps the local copies up to date as they change. If the resources are accessed without a network connection, the thin client automatically uses the local copies.

The Caching API is made of:

– the Web Storage Specification: includes an API for client-side storage of session-specific data and an API for storing session-spanning data;
– the Web SQL Database: a client-side JavaScript database;
– Web Workers: to execute parallel "background" processes in the client.

When the resource is recalled, the thin client checks to see if the manifest has changed. If so, all the necessary resources will be downloaded again. The cache will be updated only if the files that are registered with it have changed. To trigger an update, the contents of the manifest file must be changed. In Fig. 7 is presented the updated manifest in the browser console:



Manifest
/manifest.json

Identity
Add to homescreen

Name	Cross-platform responsive and progressive with offline capabilities
Short name	Progressive

Presentation

Start URL	/
Theme color	■ #2b2284
Background color	■ #2b2284
Display	standalone

Icons
72x72
image/png
Web Thin Client

Fig. 7. Browser console – thin client manifest file.

To increase maintainability and testability, of the resources the thin client under consideration was built to separate the markup and JavaScript code using the Model-View-ViewModel (MVVM), also known as Model View Presenter. The MVVM pattern splits the application into three parts Model, View and ViewModel [18] as shown below in Fig. 8.

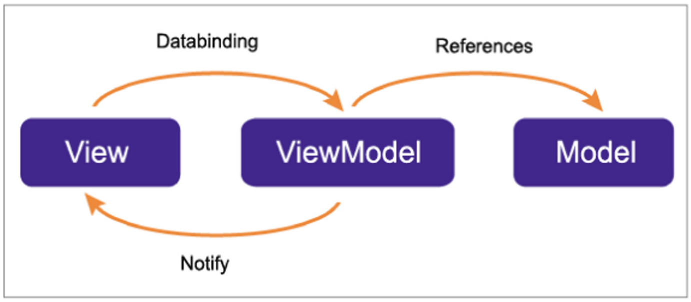


Fig. 8. Model-View-ViewModel pattern.

An important feature of a ViewModel is that it does not know the View. The bridge between the two concepts is realized via data binding, fields in the View are bounded to properties of the ViewModel. The same applies to events that occur in the View, such as clicks on buttons and operations in the ViewModel. The data binding mechanism handles both the updating of GUI (Graphical User Interface) [19] elements in the View, as the associated properties in the ViewModel change, and the transfer of user changes from the View to the ViewModel.

With the help of a Service Worker, the web application was configured to use priority cached assets, allowing for a specific user experience online even before loading more data from the network [20].

Technically, Service Workers provide a network proxy implemented with JavaScript script in the web browser to manage Web/HTTP requests from a program, interposing themselves between the network connection and the terminal providing the content. In this way cache mechanisms can be used efficiently and allow error-free behavior during long periods of offline use.

The architecture of the proposed web based thin client is presented, in Fig. 9.

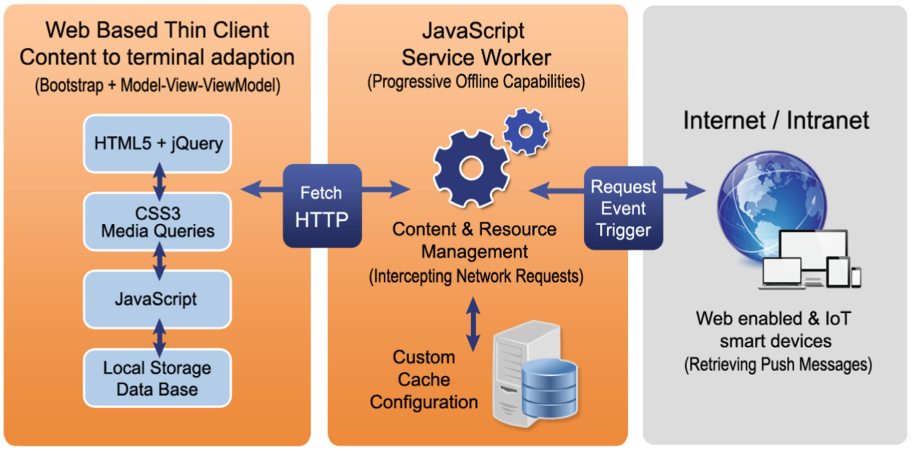


Fig. 9. Proposed architecture.

A web-enabled smart device is accessing the URLs, then the application Service Worker downloads the resources, caches them locally, and automatically keeps the local copies up to date as they change. When the resource is recalled, a request event is triggered to see if the manifest has changed. The cache will not be updated if only files that are registered with it have changed. The Cross-platform content to terminal adaption is generated using Bootstrap Framework, HTML5, jQuery, JavaScript and CSS3 Media Queries with the Model-View-ViewModel pattern.

The thin client subsequently carries out all the tasks that are assigned to it independently. Offline accessibility is managed by the Service Worker, which once installed in the navigation browser intercepts network requests and performs appropriate actions depending on whether the network is available or not.

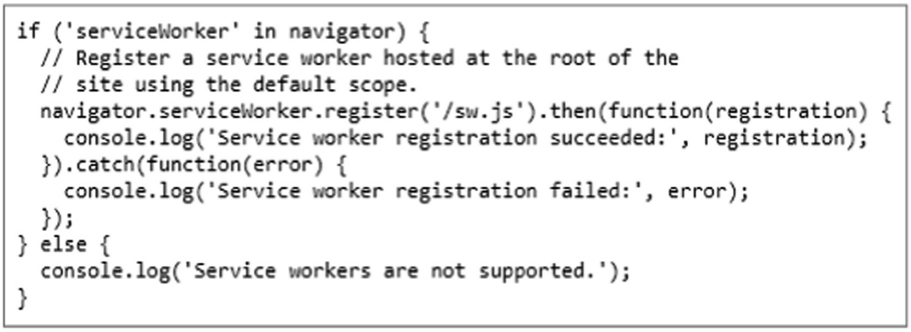
This feature allows the user to access the resources even without connection and also to improve the Quality of Experience (QoE) and Quality of Service (QoS). Even if there is a connection, some files will not need to be uploaded from the server since they have already been stored locally.

In order that JavaScript can be executed, it has always been a prerequisite that the resource was opened in a browser - using this method, on the other hand, allows a JavaScript file to be executed even if the associated resource is not open at all.

The script also has the option of loading new content in the background - for example, a previous executed measurement result. If the data are accessed at a later time, the content will be already available, as a back-up.

In addition, the thin client only works over secure HTTPS connections [21]. This has security reasons in the first place, but it was also build in a farsighted mode, because it's likely that new standards will impose that resources to sensitive features or hardware will only run over HTTPS connections in the future [22].

As previously mentioned, the JavaScript file is detached from the actual web application, and runs in the background - invisible to the visitor - and it is registered by the thin client in the first online access of the resources. This is done through the Service Worker API's "register()" method, as shown in Fig. 10.



```
if ('serviceWorker' in navigator) {
  // Register a service worker hosted at the root of the
  // site using the default scope.
  navigator.serviceWorker.register('/sw.js').then(function(registration) {
    console.log('Service worker registration succeeded:', registration);
  }).catch(function(error) {
    console.log('Service worker registration failed:', error);
  });
} else {
  console.log('Service workers are not supported.');
}
```

Fig. 10. Service worker API's - JavaScript "register()" method.

The Service Worker also runs only in its own thread, does not allow direct manipulation of the parent's DOM (Document Object Model) [23], and has the message-based interface [24]. It acts as a controller, proxy or interceptor: it has its own cache and can switch between every outgoing network request [25]. The offline capability is realized by an automatic preconfigured decision if an answer can be requested from the cache or forward the request to the network.

5 Conclusions and Future Work

The novelty and the biggest advantage of the presented web based thin client solution over classic clients is the easier operation, with effectively reduced overhead, only running the software needed to access centrally operated applications. It can operate consistently regardless of the applications that are actually being used, in accordance with the balanced Edge Computing/Cloud Computing paradigm. This also allows a very simple resource management of centralized or decentralized control systems and has a large spectrum of potential applications, being able to perform from tele-measurement tasks to management and rich context information centralization. The web based cross-platform thin client enhanced with content to terminal adaption does not need to be installed on the device - this has enormous advantages, as most of the equipment is quickly reaching the limits of storage space. Since the presented solution does not need to be installed, the operators are also independent of commercial software stores that would take shares for marketing. Because it is developed with standardized technologies, also a big asset it that it has an increased security and reduced computing power needs because the thin client does not imply third party plugins or additional dependencies which increase the risks of security breaches and often require additional resource allocation.

The architecture of our solution was presented together with the development environment and its advantages, being oriented towards achieving a decoupling of developments in the still relatively young market of mobile devices - thus, it is also likely to run on future devices. In addition, concurrent access is achieved, as an almost unlimited number of thin clients can be managed by simply assigning configurations. Quickly turning resource events and handlers on and off results in a significant service advantage for the end user, especially for remote clients and lengthy installations.

The capability of the web-based thin client solution to function offline is a main advantage, especially for companies with users who occasionally have a bad Internet connection - this is an elementary aspect because in this scenario the software consistently adopts an offline first approach. Push notifications are also available, allowing users the same access to their personal interface, configuration, directories, and installed programs, regardless of which physical thin client workstation they log on to. The endpoint management is optimized because on the thin client is running only the software necessary for server communication.

Future work will be devoted to enhancing the proposed solution, helping the user to work in a different workplace every day without any restrictions, using local storage and local databases, creating also the back-up systems in a distributed environment.

Acknowledgement. This paper has been supported by the Autonomous Region of Sardinia-Italy (Piano Sulcis Research Grant SULCIS-821101).

References

1. Oluwatosin, H.S.: Client-server model. IOSR J. Comput. Eng. **16**(1), 67–71 (2014). p-ISSN 2278-8727
2. Kim, J., Baratto, R.A., Nieh, J.: pTHINC: a thin-client architecture for mobile wireless web. In: Proceedings of the 15th International Conference on World Wide Web (WWW 2006), pp. 143–152. ACM, New York (2006)

3. Tian, Y., Song, B., Huh, E.: Towards the development of personal cloud computing for mobile thin-clients. In: 2011 International Conference on Information Science and Applications, Jeju Island, pp. 1–5 (2011)
4. Al-Hammouri, A., Al-Ali, Z., Al-Duwairi, B.: ReCAP: a distributed CAPTCHA service at the edge of the network to handle server overload. Trans. Emerg. Telecommun. Technol. **29** (4), e3187 (2017)
5. Stelea, G.A., Fernoaga, V., Gavrila, C., Robu, D.: Web-service based thin client for tele-measurement. Int. Sci. Conf. eLearn. Softw. Educ. **2**, 128–134 (2018)
6. Pohja, M.: Comparison of common XML-based web user interface languages. J. Web Eng. **9** (2), 95–115 (2010)
7. Charland, A., Leroux, B.: Mobile application development: web vs native. Commun. ACM **54**(5), 49–53 (2011)
8. Mahmood, A., Casetti, C., Chiasserini, C.F., Giaccone, P., Harri, J.: Efficient caching through stateful SDN in named data networking. Trans. Emerg. Telecommun. Technol. **29** (1), e3271 (2010)
9. Richards, G., Lebresne, S., Burg, B., Vitek, J.: An analysis of the dynamic behavior of JavaScript programs. SIGPLAN Not. **45**(6), 1–12 (2010)
10. Vaughan-Nichols, S.J.: Will HTML 5 restandardize the web? J. Comput. **43**(4), 13–15 (2010). https://doi.org/10.1109/MC.20
11. Orsini, G., Bade, D., Lamersdorf, W.: CloudAware: empowering context-aware self-adaptation for mobile applications. Trans. Emerg. Telecommun. Technol. **29**(4), e3210 (2017)
12. Rajesh, N.A.: Responsive web design. Int. J. Eng. Comput. Sci. **4**(3)
13. Robu, D., Fernoaga, V., Stelea, G.A., Sandu, F.: Tele-measurement with virtual instrumentation using web-services. Des. Technol. Electron. Packag. (SIITME) **23**, 387–394 (2017)
14. Baturay, M.H., Birtane, M.: Responsive web design: a new type of design for web-based instructional content. Procedia – Soc. Behav. Sci **106**, 2275–2279 (2013). ISSN 1877-0428
15. Joorabchi, M.E., Mesbah, A., Kruchten, P.: Real challenges in mobile app development. In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 2851–2864 (2013)
16. Smutny, P.: Mobile development tools and cross-platform solutions. In: Proceedings of the 13th International Carpathian Control Conference (ICCC), pp. 653–656 (2012). https://doi.org/10.1109/carpathiancc.2012.6228727
17. Maddah-Alim, M.A., Niesen, U.: Fundamental limits of caching. IEEE Trans. Inf. Theory **60**, 2856–2867 (2014). https://doi.org/10.1109/TIT.2014.2306938
18. Leff, A., Rayfield, J.T.: Web-application development using the Model/View/Controller design pattern. In: Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference, pp. 118–127 (2001). https://doi.org/10.1109/edoc.2001.950428
19. Banerjee, I., Nguyen, B., Garousi, V., Memon, A.: Graphical user interface (GUI) testing: systematic mapping and repository. J. Inf. Softw. Technol. **55**(10), 1679–1694 (2013). ISSN 0950-5849
20. Bhamare, D., Samaka, M., Erbad, A., Jain, R., Gupta, L.: Exploring microservices for enhancing internet QoS. Trans. Emerg. Telecommun. Technol. **29**, e3445 (2018)
21. Clark, J., Oorschot, P.C.: SoK: SSL and HTTPS: revisiting past challenges and evaluating certificate trust model enhancements, pp. 511–525 (2013)
22. Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., Shmatikov, V.: The most dangerous code in the world: validating SSL certificates in non-browser software. In: CCS 2012, pp. 38–49. ACM, New York (2012)

23. Jensen, S.H., Madsen, M., Moller, A.: Modeling the HTML DOM and browser API in static analysis of JavaScript web applications. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 59–69 (2013). https://doi.org/10.1145/2025113.2025125
24. Arnbak, A., Asghari, H., Van Eeten, M., Van Eijk, N.: Graphical user interface (GUI) testing: systematic mapping and repository. Commun. ACM **57**(10), 47–55 (2013). https://doi.org/10.1145/2660574
25. Leu, J., Chen, C., Hsu, K.: Improving heterogeneous SOA-based IoT message stability by shortest processing time scheduling. IEEE Trans. Serv. Comput. **7**(4), 575–585 (2014)