

# Appendix of “Energy Patterns for Web: An Exploratory Study”

Pooja Rani  
rani@ifi.uzh.ch  
University of Zurich  
Zurich, Switzerland

Jonas Zellweger  
jonas.zellweger@uzh.ch  
University of Zurich  
Zurich, Switzerland

Veronika Kousadianos  
veronika.wu@students.unibe.ch  
University of Bern  
Bern, Switzerland

Luis Cruz  
Delft University of Technology  
Delft, The Netherlands

Timo Kehrer  
University of Bern  
Bern, Switzerland, Switzerland

Alberto Bacchelli  
University of Zurich, Switzerland  
Zurich, Switzerland

## ABSTRACT

As the energy footprint generated by software is increasing at an alarming rate, understanding how to develop energy-efficient applications has become necessary. To develop energy-efficient code, previous work has introduced catalogs of coding practices, also known as energy patterns. These patterns are yet limited to Mobile or third-party libraries. In this study, we focus on the Web domain—a main source of energy consumption. First we investigated whether and how Mobile energy patterns can be ported to this domain and found that 20 patterns could be ported. Then, we interviewed six expert web developers from different companies to challenge the ported patterns. To quantify the effect of Web energy patterns on energy consumption, we set up an automated pipeline to evaluate two ported patterns: ‘Dynamic Retry Delay’ (DRD) and Open Only When Necessary (OOWN). In this document, we describe the steps to obtain the artifact package of our study, the artifact material, and how to use the artifact.

### ACM Reference Format:

Pooja Rani, Jonas Zellweger, Veronika Kousadianos, Luis Cruz, Timo Kehrer, and Alberto Bacchelli. 2023. Appendix of “Energy Patterns for Web: An Exploratory Study”. In *Proceedings of 46th International Conference on Software Engineering (ICSE 2024)*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

## 1 RQ<sub>1</sub> ADAPTATION OF ENERGY PATTERNS

For each pattern, we provide the original definition given by Cruz *et al.*, the adapted context for web applications, a solution, and an example [37]. The discussion part of each section can provide comparisons between mobile and web applications, examples, and research from other domains for that pattern. If the pattern only applies to a certain extent or in certain situations, the discussion aims to present the reasons and explanations.

### 1.1 Dark UI Colors

**Original:** “Provide a dark UI color theme to save battery on devices with AMOLED screens [37].”

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

ICSE 2024, April 2024, Lisbon, Portugal

© 2023 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

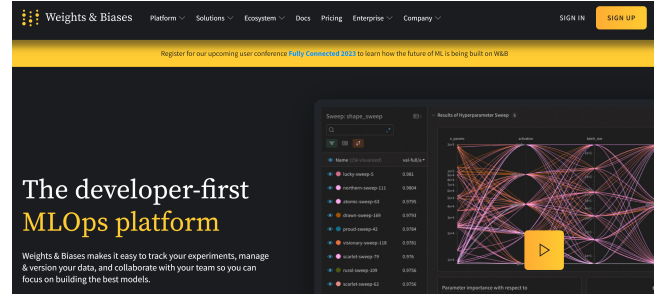


Figure 1: Screenshot of a website with dark UI colors.

### Adaptation to Web.

**Context:** Dark mode can be applied to anything with a GUI (graphical user interface - also sometimes referred to only as UI user interface). Since websites provide interactive GUIs, dark background colors can be applied to web UIs.

**Solution:** Provide a UI with dark background colors.

**Example:** There already exist websites that provide dark UI colors like wand.ai (see Figure 1 [26]).

**Discussion:** In mobile devices, the display alongside the GSM module are the two components with the highest power consumption [30]. Providing a dark UI color theme on devices with OLED (Organic Light Emitting Diodes) screens has been shown to reduce the display’s power consumption up to 83%, which can translate into a 69% total phone power reduction [39]. This indicates that switching from light to dark mode saves a significant amount of battery. The percentage of global web traffic recorded on mobile phones has increased drastically over the past decade. In February 2023, 60.67% of all web traffic came through mobile phones [20]. Therefore, we expect similar benefits to battery life when accessing a web page with dark UI colors from a smartphone. To our knowledge, no research exists on energy consumption reduction of dark UI colors in non-battery-powered devices. We assume that depending on the monitor type used to display the web pages - namely, the newly emerging OLED monitors - there will be a reduction of energy consumption. Generally, we see dark UI colors catching on in web applications with motivations not only restricted to reducing power consumption: they range from adjusting brightness to current lighting conditions and improved display readability in dark environments [49] [57], to having beneficial impact on users with reduced vision in certain circumstances [27] and of course reduce power consumption on battery-powered devices [42]. Given

**Table 1: Mobile energy patterns by Cruz *et al.* [37] with their description**

Pattern	Description
Dark UI Colors	Provide a dark UI color theme to save battery on devices with AMOLED screens.
Dynamic Retry Delay	Whenever an attempt to access a resource fails, increase the time interval before retrying to access the same resource.
Avoid Extraneous Work	Avoid performing tasks that are either not visible, do not have a direct impact on the user experience to the user or quickly become obsolete.
Race-to-idle	Release resources or services as soon as possible (such as wake locks, screen).
Open Only When Necessary	Open/start resources/services only when they are strictly necessary.
Push over Poll	Use push notifications to receive updates from resources, instead of actively querying resources (i.e., polling).
Power Save Mode	Provide an energy efficient mode in which user experience can drop for the sake of better energy usage.
Power Awareness	Have a different behavior when the device is connected/disconnected to a power station or has different battery levels.
Reduce Size	When transmitting data, reduce its size as much as possible.
WiFi over Cellular	Delay or disable heavy data connections until the device is connected to a WiFi network.
Suppress Logs	Avoid using intensive logging.
Batch Operations	Batch multiple operations instead of putting the device into an active state many times.
Cache	Avoid performing unnecessary operations by using cache mechanisms.
Decrease Rate	Increase time between syncs/sensor reads as much as possible.
User Knows Best	Allow users to enable/disable certain features in order to save energy.
Inform Users	Let the user know if the app is doing any battery-intensive operations.
Enough Resolution	Collect or provide high-accuracy data only when strictly necessary.
Sensor Fusion	Use data from low-power sensors to infer whether new data needs to be collected from high-power sensors.
Kill Abnormal Tasks	Provide means of interrupting energy greedy operations (e.g., using timeouts or user input).
No Screen Interaction	Whenever possible, allow interaction without using the display.
Avoid Extraneous Graphics and Animations	Graphics and animations are really important to improve the user experience. However, they can also be battery intensive ? use them with moderation.
Manual Sync – On Demand	Perform tasks exclusively when requested by the user.

this pattern’s various benefits, it would be interesting to survey users to gather other possible benefits.

## 1.2 Dynamic Retry Delay

**Original:** “Whenever an attempt to access a resource fails, increase the time interval before retrying to access the same resource [37].”

### *Adaptation to Web.*

**Context:** The concept of managing retries to access a previously failed resource is a common topic across all computer science domains, including web applications. For instance, in client-server-based applications, the front and backend communication frequently happens through APIs. Specifically, the frontend sends a request to the backend, and based on the availability of the resource, the backend replies to the request.

**Solution:** Use a heuristic (e.g., linear or exponential increase) to manage the retry interval after each failed attempt to connect.

**Example:** Consider a web application that displays a news feed. The frontend relies on the backend to provide updates. In case the frontend can not reach the backend, it can either continuously send requests to the backend or can increase the time between attempts using some sort of heuristic (e.g., linear or exponential growth). Listing 1 shows an implementation utilizing the Fibonacci series to delay the retry attempts.

```
val retryStrategy =
  RetryStrategy.fibonacciBackOff(initialWaitDuration = 3.
    seconds, maxAttempts = 5)
```

**Listing 1: Example heuristics using the fibonacci series [56]**

**Discussion:** Research concerning the impact of energy consumption in connection with (re-)accessing resources is being conducted in different subdomains of computer science already. For example, in Wireless Sensor Networks (WSN), energy consumption is one of the main design constraints alongside scalability, delay, traffic adaptation, throughput, and overheads. Therefore, an efficient Medium Access Control (MAC) protocol is required [46]. Cho *et al.* proposed a back-off algorithm for MAC protocols in WSN that uses a dynamic contention period based on the remaining energy (i.e., battery level) of each sensor node [33]. They showed that this algorithm saves more power and prolongs the lifetime of sensor nodes compared to conventional back-off algorithms. We assume that using this and similar back-off algorithms would also decrease power consumption in other contexts, not only in wireless sensor networks.

A direct consequence of re-accessing a resource is that the end user experiences a delay while interacting with the application. Whenever a user experiences a delay in communication over a network, we speak of latency. Network latency can be described as the time it takes for a data package to travel from one point in the network to another. There exists no agreement on what a good latency is. For example, an acceptable latency in gaming is around 40 - 60 milliseconds or less [1], while web domains accept values below 150 milliseconds for loading websites [2] and below 100 milliseconds when streaming videos or video conferencing [3]. This implies that as developers, we must be aware of the web application’s purpose and aim for the maximum latency. While measuring the energy impact, we need to consider the sources, the domain, and the medium through which the internet is accessed as the latency is significantly lower when using an Ethernet connection compared to Wi-Fi [7].

### 1.3 Avoid Extraneous Work

**Original:** "Avoid performing tasks that are either not visible, do not have a direct impact on the user experience or quickly become obsolete [37]."

*Adaptation to Web.*

**Context:** Web applications can perform a number of tasks simultaneously. There are cases in which the result of those tasks is either not visible (e.g., the UI is presenting other pieces of information), or the result is not necessarily relevant to the user. This is particularly critical when the website goes to the background (e.g., the user changes to another tab in the browser).

**Solution:** Investigate the relevant set of data to be presented to the user in each use case. Enable/disable updates and processing tasks depending on their effect on the visible or valuable data to the user.

**Example:** Consider the code snippet in Listing 2. It shows an example where a website pauses audio when the user switches to a different tab and plays when they switch back. In this case, no unnecessary energy consumption will occur when the user is not interacting with the page.

**Discussion:** For web applications, for example, there exists the Page Visibility API [36]. When the user switches to another tab or minimizes the window, this API sends an event to let listeners know that the state of the page has changed. This is especially useful for improving performance and also saving resources by not letting a page perform unnecessary tasks when the document isn't visible. See also the discussions in 1.5 Open Only When Necessary and in 1.4 Race-to-idle.

In the context of web applications, its data, in the form of content, does not necessarily become as 'quickly obsolete' as in mobile applications. For example, recommendations based on location e.g., for restaurants or for ATMs, will change if the user is on the move. Therefore, the location data from a couple of minutes ago can become obsolete. If we implement a mobile application, we expect the user to be on the move frequently. But if we implement a web application, we can not generally expect this behavior. Therefore, we recommend setting the focus of this pattern in web applications on tasks that are either not visible or do not directly impact the user experience.

```
const audio = document.querySelector("audio");

// Handle page visibility change:
// - If the page is hidden, pause the video
// - If the page is shown, play the video
document.addEventListener("visibilitychange", () => {
  if (document.hidden) {
    audio.pause();
  } else {
    audio.play();
  }
});
```

**Listing 2: Code snippet from Mozilla API that handles page visibility change [36]**

### 1.4 Race-to-idle

**Original:** "Release resources or services as soon as possible (such as wake locks, screen) [37]."

*Adaptation to Web.*

**Context:** Releasing resources and services that are no longer needed is a very sensible coding practice that can be applied anywhere in computer science, including in web applications. The resources in the context of web applications refer to the backend, external servers, databases, and device hardware (e.g., camera, microphone, and speaker).

**Solution:** Release unnecessary resources and services. This includes releasing the hardware resources that are no longer used (e.g., camera after a video call) and applying resource management patterns (e.g., garbage collection) during software development.

**Example:** Soundcloud uses React Redux in their techstack, which includes the functionality of garbage collection [50]. **Discussion:** There exist a number of resource management patterns in software development. For instance, Resource manager patterns [43] for embedded and real-time systems utilize lists to keep track of free and busy resources to figure out how to allocate and free up resources in the system. Another common pattern is the dispose pattern [10], which is used across many object-oriented programming languages. Here, resources held by objects are released by calling methods like close(), release(), etc. This pattern is primarily used in those languages whose runtime environment have automatic garbage collection, such as C# and Java. Other patterns regarding releasing resources include the *Leasing pattern* and the *Evictor pattern* [4].

For web application development, resource management patterns can be used in both front- and backend. According to intelipaat.com [13], the 4 most popular languages for backend development in web applications are Python, php, Java, and C#, all of which support garbage collection. The most used language for the frontend is JavaScript along with its frameworks including Angular, React, and VueJS. JavaScript also supports garbage collection as a form of memory management [5].

Browsers can promote similar behavior in websites themselves. WebKit [23], a web browser engine used by Safari and other apps on macOS, iOS, and Linux, automatically displays the following behavior when a page becomes inactive:

- window.requestAnimationFrame() is stopped,
- CSS and SVG Animations are halted,
- Timers are suspended <sup>1</sup>.

### 1.5 Open Only When Necessary

**Original:** "Open/start resources/services only when they are strictly necessary [37]."

*Adaptation to Web.*

**Context:** In web applications, some resources require to be opened before using them. In addition, there exist services and or resources that improve user experience but are non-critical. It is possible to open/load these resources and services at the very start when accessing the web page for the first time. However, the resources will actively wait for events or requests and passively consume energy even though they are not in use.

**Solution:** When accessing hardware, open resources only when necessary. During software development, identify non-critical services and load those only when necessary.

<sup>1</sup><https://webkit.org/blog/8970/how-web-content-can-affect-power-usage/>

**Example:** A web application opens the camera only when necessary for capturing an image - not earlier when the application has been started (see Listing 4). Another example can be found in Listing 3. It shows how images can be loaded lazily in web applications.

```
img src="image.png" loading="lazy" alt="..." width="200" height="200"
```

**Listing 3: Code snippet showing the usage of the loading attribute to lazy-load images [41]**

**Discussion:** Various hardware resources are available to websites, e.g., a camera, microphone, and speaker. There exist APIs, e.g., the Media Capture and Streams API (Media Stream) [21], which allow developers to manipulate audio- or video-related data, including the access of said hardware.

In frontend development, there exists a widely used pattern called lazy loading. It allows for identifying non-critical resources and loading them only when necessary, avoiding unnecessary energy consumption. In addition, Tuaycharoen *et al.* show that lazy loading can increase the performance of web applications by 50% due to an improved response time [60]. The web application can use lazy loading in different scenarios and places. One typical use case happens during user interactions, such as scrolling and navigating.

```
<template>
  ...
  <div class="camera-icon" @click="captureImage">
    <i class="big camera icon" ></i>
    <p>Take Picture</p>
  </div>
  ...
</template>

<script>
  ...
  methods: {
    captureImage() {
      const mediaStreamTrack = this.mediaStream.
        getVideoTracks()[0]
      const imageCapture = new window.ImageCapture(
        mediaStreamTrack)
      let reader = new FileReader();
      return imageCapture.takePhoto().then(blob => {
        reader.readAsDataURL(blob)
        reader.onload = () => {
          this.imageData.image = reader.result;
        }
      })
    }
  }
  ...
</script>
```

**Listing 4: Accessing the camera only when actually capturing the image - not earlier [29]**

## 1.6 Push over Poll

**Original:** “Use push notifications to receive updates from resources, instead of actively querying resources (*i.e.*, polling) [37].”

*Adaptation to Web.*

**Context:** Handling updates from resources (*e.g.*, servers) is a crucial part of the web application design. Different approaches exist to get updates: the client repeatedly queries the server in set intervals to see if new data is available (*i.e.*, polling), or the server

pushes new data to the client as soon as it is available. Since it is impossible to know if the server has new data available at this moment, the client continuously polling data will inevitably lead to unnecessary querying and thus may consume more energy.

**Solution:** Use the pushing mechanism to get updates from resources.

**Example:** Incorporate push notifications into the web application where possible. Listing 5 shows a possible implementation of such a push notification.

```
public async Task SendNotificationAsync(PushSubscription
  subscription,
  PushMessage message, CancellationToken cancellationToken)
{
  try
  {
    await _pushClient.RequestPushMessageDeliveryAsync(
      subscription, message,
      cancellationToken);
  }
  catch (Exception ex)
  {
    await HandlePushMessageDeliveryExceptionAsync(ex,
      subscription);
  }
}
```

**Listing 5: Code snippet of a push notification implementation.**

**Discussion:** To avoid sending requests that will return no update in order to reduce unnecessary energy consumption is relevant for mobile applications since users rely on batteries. Ayala *et al.* found that using the polling mechanism consumes up to 10% more energy than using long polling and up to 7.5% more energy for WebSocket in Android phones [25]. When users access a web page from a mobile device, we would expect similar findings. To our knowledge, there exists no research about energy reduction comparing pulling and polling for non-mobile devices.

Web push notifications were introduced around 2015 by Google and Mozilla for Chrome 48 and Firefox 44, respectively.<sup>2</sup> They are widely used in e-commerce and social media to keep users engaged [9]. Downsides of push notifications for mobile applications have been researched by Kollmann *et al.* [47]. The main concern is privacy since crucial information like the user’s location (*i.e.*, IP address) is shared with the notification service. Similar issues could possibly affect web push notifications.

## 1.7 Power Save Mode

**Original:** “Provide an energy efficient mode in which user experience can drop for the sake of better energy usage [37].”

*Adaptation to Web.*

**Context:** Providing energy efficient modes in which some functionalities are restricted or even dropped entirely can be implemented for any application (mobile and web).

**Solution:** Offer an energy-efficient mode in which the application consumes less energy.

**Example:** BooHoo has introduced an energy-saving mode to its website (see Figure 2).

<sup>2</sup><https://www.digitalmarketer.com/blog/use-web-push-notifications/>



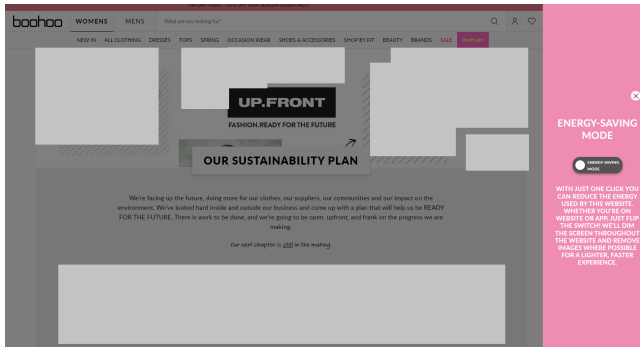


Figure 2: Screenshot of a website with energy-saving mode [40].

**Discussion:** Providing energy-efficient modes in which some functionalities are restricted or even dropped entirely can be implemented for any mobile or web-based application. Using such an energy-efficient mode should reduce energy usage in any scenario. However, unless the user is on a device powered by a battery, they are unlikely to notice a reduction in energy consumption.

Web applications are accessed via browsers. While browsers are not web applications themselves, they can play a crucial role in controlling the energy consumption of websites by potentially allowing web applications to provide energy-saving modes. As of February 2023, Google Chrome has been reported to perform testing on functionality that allows websites to switch to battery-saving mode [53] [44]. The browser Opera offers a battery-saving mode that reduces energy consumption when browsing any website, even in non-battery powered devices [17]. We conclude that there is a growing awareness of power-saving modes in the web application domain.

## 1.8 Power Awareness

**Original:** "Have a different behavior when the device is connected/disconnected to a power station or has different battery levels [37]."

*Adaptation to Web.*

**Context:** If web applications are used by devices without battery this pattern is not applicable. However, assuming the device used to access the web application has a battery, there are ways of adapting the behavior of an application according to the battery level or if the device is connected/disconnected.

**Solution:** Enable or disable certain functionalities of the device according to its power status.

**Example:** VueUse, a collection of Vue composition utilities, provides a reactive Battery Status API [22], which gives information about the system's battery level and charging status. For devices without battery, the boolean 'charging' will always be true.

**Discussion:** Websites can access information about the system's battery charge level through various APIs, for example, framework-specific APIs like VueUse's `useBattery()` or more general ones like the Battery Status API [15], which uses the built-in `window.navigator` property of the navigator object in web pages. Thus, it is possible for developers to implement different behavior depending on the

battery level of the device. Concerns have been raised that companies may utilize this information to their benefit and monetize access to battery levels. When the battery is running low, users might make different decisions than otherwise.<sup>3</sup>

Chrome has introduced a new feature called Energy Saver [31], which limits background activity and visual effects (e.g., animations and videos) as soon as the battery of the device reaches 20%. Similarly, the aforementioned battery saving mode for the Opera browser can be configured to turn on automatically when the device is removed from the power station and, when unused, will send a reminder once the battery reaches 20% that the mode is available [17].

## 1.9 Reduce Size

**Original:** "When transmitting data, reduce its size as much as possible [37]."

*Adaptation to Web.*

**Context:** Data transmission over a network is a routine operation in web applications, which can be energy greedy.

**Solution:** Transmit only what is necessary; avoid sending unnecessary data. Use data compression.

**Example:** Consider using headers *Accept-Encoding* and *Content-Encoding* for HTTP requests. Listing 6 shows an implementation where strings bigger than 1KB get compressed.

**Discussion:** Data transmission is a routine operation in web applications. Moreover, such operations can be energy greedy. The International Energy Agency (IEA) estimated the global internet traffic of 2021 to be 3.4 ZB (1 zetabyte =  $10^{21}$  bytes), which has led to an increase of up to 60% in the global data transmission network energy use since 2015 [6]. Reducing internet traffic can be achieved by compressing data before transmission. However, the work of Barr *et al.* [28] shows that in some cases, there can be a net energy increase when compression is applied before transmission. The energy consumption difference depends on hardware factors such as the relative energy of CPU, memory, and network, as well as software factors, including compression ratio and memory access patterns. In their work, the authors have measured up to 31% overall energy reduction when using adequate compressors and decompressors for sending compressed web data [28].

Not only can the data itself be compressed, but the request as a whole can be further optimized. Tools commonly used to make HTTP requests in web applications include XMLHttpRequest (XHR), JQuery Ajax, Fetch, Axios and Angular HttpClient [48]. Using these packages allows developers to edit and adapt the requests being sent in their application, which allows, for example, the removal of unnecessary headers.

<sup>3</sup>Podcast Episode 'Your Brain on Uber' on the Hidden Brain Podcast released in May 2016, accessed in March 2023

```

import axios from 'axios';
import pako from 'pako';

const api = axios.create({
  baseURL: "http://localhost:9000/api",
  withCredentials: true,
  transformRequest: axios.defaults.transformRequest.concat(
    function (data, headers) {
      // compress strings if over 1KB
      if (typeof data === 'string' && data.length > 1024) {
        headers['Content-Encoding'] = 'gzip';
        return pako.gzip(data);
      } else {
        headers['Content-Encoding'] = undefined;
        return data;
      }
    }
  )
});

```

**Listing 6: Impelemtation of data compression on strings larger than 1KB [35]**

## 1.10 Suppress Logs

**Original:** “Avoid using intensive logging [37].”

*Adaptation to Web.*

**Context:** Logging is common practice in web applications. Therefore, suppressing logs can be applied.

**Solution:** Refrain from extensive logging. Keep the logging rates below one message per second.

**Example:** The console output in Listing 7 shows an example of possible logs. The logging level is *warning*. This logging level includes warnings, errors, and critical system failures.

```

# loglevel = warning

backend | 2023-07-19 13:48:56,133 model_factory Response to
request with pk 1234-5678-abcd
has content type text/javascript but was unable to parse it

backend | 2023-07-19 13:49:02,343 model_factory Response to
request with pk abcd-1234-5678
has content type text/javascript but was unable to parse it

backend | 2023-07-19 13:49:12,876 model_factory Response to
request with pk 1234-abcd-5678
has content type text/javascript but was unable to parse it

```

**Listing 7: Console log of an interaction with a web application on log level warning. Other logging levels include warnings, errors, and critical system failures.**

**Discussion:** Shaiful *et al.* found, that execution logs for debugging can influence energy consumption in Android applications [34]. They show that the generation of logs at rates of one message per second or more negatively impacts energy consumption. Logging is one of the basic functionalities of most web applications. It records valuable information for operating and maintaining the website, such as the running information, user operations, debugging, warnings, and error information of the application [52]. To our knowledge, no study exists on the energy impact of logging in web applications. However, we assume, that extensive logging can also impact power consumption in web applications. Further, web applications are usually deployed in production mode, which by design does not support debugging. Therefore, the number of

debugging logs in deployed websites will be zero. Thus eliminating the possibility of higher energy consumption caused by debugging logs. However, there are still logs available in production mode. There exist different logging levels: critical, error, warning, info, and debug are among the most common ones [18]. The impact of continuously logging for different purposes on the server side or in source code is yet to be investigated.

## 1.11 Batch Operations

**Original:** “Batch multiple operations, instead of putting the device into an active state many times [37].”

*Adaptation to Web.*

**Context:** Batch processing is widely used in applications that contain tasks that can be executed without user interaction, also in web applications.

**Solution:** Combine multiple operations into a batch for processing. By bundling multiple tasks, the overall efficiency can be improved, and possible tail energy consumption (energy consumption related to starting and stopping resources) can be reduced.

**Example:** Use Job Scheduling APIs for managing multiple tasks.

```

def start_process():
    _create_and_return_bills()

    some_job = django_rq.enqueue(_setup, jobs,
                                job_timeout=7200)

    bill_jobs = []

    # Generate bills using workers.
    for bill in list_of_bills:
        bill_jobs.append(
            django_rq.enqueue(_generate_bill, bill,
                              some_job.id,
                              depends_on=some_job))

    # Bills for user are calculated concurrently
    bill_jobs.append(django_rq.enqueue(
        set_total_costs_for_billing,
        all_bills, depends_on=bill_jobs))
    django_rq.enqueue(concat_files, all_bills,
                      depends_on=bill_jobs)
    ...

```

**Listing 8: Usage of job scheduling in implementation to be able to batch operations [38]**

**Discussion:** One concern when executing operations separately on mobile phones is tail energy consumption [37]. When batching multiple tasks, tail energy consumption can be optimized since an active state can be maintained until all tasks are done and do not need to be re-activated for each operation individually. Background tasks, such as managing cellular connection, can be expensive but do not necessarily have to be executed in a specific time. Cruz *et al.* proposed to use Job scheduling APIs (e.g., `android.app.job.JobScheduler`) for managing multiple background tasks occurring in a device. Job scheduling is also available for web applications. One example is the Google Cloud Scheduler API [11], which allows the scheduling of multiple tasks by use of cron jobs.

For web application development, there exist batch processing frameworks e.g., batch processing framework in Java EE [14]. This framework executes tasks periodically or when resource usage is low. Common use cases for batch processing include billing,

report generation, data format conversion, and image processing. Developers can even be mindful of batch operations during frontend development: when using the REST API, it is possible to make use of the batch mode, which allows the bundling of multiple individual requests into one larger request [59] [38].

## 1.12 Cache

**Original:** "Avoid performing unnecessary operations by using cache mechanisms [37]."

*Adaptation to Web.*

**Context:** Caching is widely adapted and used in web applications. Websites routinely collect data from the server. Browsers, for example, provide storage that can be utilized for caching.

**Solution:** Utilize caching mechanisms to reduce network load.

**Example:** Utilize local storage provided by the browser to avoid collecting the same data repeatedly. Listing 9 shows a code example for the frontend on how to cache an API response in the local storage.

```
let data = localStorage.getItem('data')
axios.get('/data').then((response) => {
  data = response.data.data
  localStorage.setItem('data', response.data.data)
})
```

**Listing 9: Code snippet of a push notification implementation [54].**

**Discussion:** Web caching is a well-known strategy that stores popular web objects that are likely to be visited in the near future in positions close to the user. Thus, web caching helps reduce network traffic. This mechanism can be implemented at three different levels: client level (browser cache), proxy level (proxy server cache), and original server level (cache server) [24]. Browser caching is located on the client side. Modern web browsers provide cache mechanisms, which are especially useful when a user wants to return to a page they have recently looked at. Proxy server cache is similar to browser caching but works on a much larger scale. When a request arrives at the proxy server, it checks its cache. If the desired object is available, the proxy sends the object to the client. If the object has expired or is unavailable, the proxy server sends a request to the original server and passes that response to the client. This object will also be stored from the proxy in its own cache for future requests. The last level on which caching mechanisms are commonly implemented is on the original server. This server-side caching aims to reduce the need for redundant computations or database retrievals [24]. Traditional methodologies for implementing web caching are based on time and size. They do not consider the events that will possibly happen in the future. Pernabas *et al.* propose to improve the performance of web proxy caching by adapting the Data Mining Classifier model based on Web User clustering and Weighted Random Indexing Methods [55].

## 1.13 Decrease Rate

**Original:** "Increase time between syncs/sensor reads as much as possible [37]."

*Adaptation to Web.*

**Context:** In web applications, data synchronization usually happens through requests. How often these requests are sent is determined by the architecture of the application and the loading mechanism chosen by the developers.

**Solution:** Increase the time between requests to the backend as much as possible as long as it does not harm performance or user experience. Consider implementing dynamic or lazy loading. Examine the possibility of using pushing mechanisms over polling.

**Example:** Consider a web application that shows the availability of books in a library. Updating the availability several times throughout the day will be accurate enough for this use case. There is no need to synchronize and update multiple times per minute.

**Discussion:** See Discussion in 1.6 Push Over Poll and in 1.5 Open Only When Necessary.

## 1.14 User Knows Best

**Original:** "Allow users to enable/disable certain features in order to save energy [37]."

*Adaptation to Web.*

**Context:** Implementations aiming for energy efficiency often come with a trade-off between features and power consumption [37]. Different users accept different trade-offs. For example, some users might tolerate the absence of certain features for better energy efficiency and vice versa.

**Solution:** Allow for customization of preferences by the end user regarding energy-critical features.

**Example:** With the energy-saving mode on their website, BooHoo [40] allows users to choose between certain features (bright screen and full availability of images) and energy consumption (see Figure 2).

**Discussion:** See Discussion in 1.7 Power Save Mode.

## 1.15 Inform Users

**Original:** "Let the user know if the app is doing any battery intensive operations [37]."

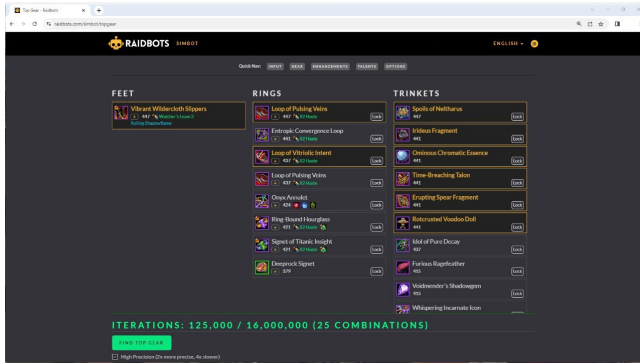
*Adaptation to Web.*

**Context:** In web applications with client-server architecture, computation-heavy operations usually happen in the backend. The battery level of the device through which the user is accessing the web page does not have any impact on these operations. While it is technically possible to alert users to power-intensive operations in the backend, the impact on user experience remains to be researched.

**Solution:** Give users informative feedback when the web application performs power-intensive operations.

**Example:** Imagine a web application that simulates character strength for a video game based on gear. The user can choose different items (*i.e.*, weapon or armor), and the web application iterates over all possible combinations to find the best one. Raider.io [19] already provides the number of combinations that will be simulated. Information about the energy consumption could easily be added as well *e.g.*, low, average, or high consumption.

**Discussion:** In HCI (Human Computer Interaction), Shneiderman has proposed eight golden rules [8]. 'Offer Informative Feedback' proposes offering users feedback during or immediately after



**Figure 3: Website showing the number of all possible combinations of the gear selected by the user [19].**

an action. Such informative feedback could easily be extended with remarks about power consumption. Since power-intensive operations usually happen on the server, this does not influence the power consumption of the clients' devices. The impact of this pattern on user experience remains to be researched.

## 1.16 Enough Resolution

**Original:** "Collect or provide high accuracy data only when strictly necessary [37]."

*Adaptation to Web.*

**Context:** Web pages routinely collect data from web servers and provide and display this data to the user. Therefore, it is possible for web applications to collect and provide high-accuracy data only when necessary.

**Solution:** Study the importance of the collected and provided data to the user experience. Find the optimal balance between quality and size.

**Example:** Consider a game web application. High resolution and high frame rates are necessary during game play, but when the user is in the menu, using lower frame rates is possible without reducing the overall user experience.

**Discussion:** Collection and manipulation of high-resolution data requires more resources (e.g., memory, processing, capacity, etc.) than when using low resolution [37]. Thus, the energy consumption is higher when processing high-resolution data. Greenspector, a measurement and analysis solution to reduce the environmental impact of mobile and web applications, has performed a study investigating the energy impact of different image formats and resolutions on websites [12]. They found that energy consumption can be reduced by a factor of 2 for image qualities that remain high after compression and by a factor of 6 for compression to low-quality images. Chiasserini *et al.* [32] conducted a study comparing the overall energy consumption of transmitting compressed versus non-compressed videos and still images. They show that, for example, compression of JPEG images with integer DCT kernel reduces energy costs compared to non-compressed images.

Another idea of applying this pattern to web applications could be to store, for example, several versions of the same image in different resolutions in the backend. Then the frontend can request

the appropriate level of resolution depending on the use case and need not request a high-resolution image every time.

## 1.17 Kill Abnormal Tasks

**Original:** "Provide means of interrupting energy greedy operations (e.g., using timeouts, or users input) [37]."

*Adaptation to Web.*

**Context:** Timeouts are used frequently in web applications. They can be used in many different contexts. For example, session timeouts are used to handle the situation when a user does not perform any action on a website during a set interval (determined by the web server). Additionally, it is possible to offer manual means of interruption. As stated in the context of 1.15, one of Shneiderman's eight golden rules proposes to give users informative feedback immediately after an action has been performed. It is possible to inform users of power-intensive operations and, in the same dialog, offer means of manually interrupting those operations.

**Solution:** Provide means of interrupting energy greedy operations.

**Example:** Implementation of a timeout can be seen in Listing 10. Here, a timeout has been set to manage the calculation of different values needed to create a bill. If something goes wrong during the calculations, the timeout ensures that the abnormal behavior gets interrupted and that the resource (in this case, a worker) will be freed up again.

```
some_job = django_rq.enqueue(_setup, jobs,
                             job_timeout=7200, result_ttl=1800)
```

### Listing 10: Example of a timeout.

**Discussion:** The main concern for this pattern in mobile phones is the battery level of the device [37]. In web applications, energy-intensive operations typically happen in the backend. This does not impact the energy consumption of the system through which the end user is accessing the web page. However, timeouts are already being used in web applications. One frequent use case is associated with a WebDriver session. Here, various timeouts control behaviors for script injection, document navigation, and element retrieval [16]. Additionally, it is conceivable for developers to inform users about computation-heavy operations (see Discussion in 1.15). In HCI, ten usability heuristics by Nielsen exist [51]. One of which addresses user control and freedom. This heuristic states that it should always be possible for users to exit the current interaction with, for example, a cancel button. By following this suggestion, developers can ensure that any kind of interaction or task can be interrupted. Therefore, providing a means of interrupting especially energy-greedy operations would follow the usability heuristics by Nielsen.

## 1.18 No Screen Interaction

**Original:** "Whenever possible allow interaction without using the display [37]."

*Adaptation to Web.*

**Context:** Most websites require the continuous usage of the screen. There exist a few use cases where interactions can happen



without using the screen (e.g., allowing keyboard input to pause/start videos or to adjust volume). However, shutting down the screen while actively using a website seems only applicable to a limited number of use cases.

**Solution:** Allow users to interact with the web application using alternative interfaces (e.g., keyboard input, audio).

**Example:** A website that provides an audio player could allow the screen to turn off while playing music. As an alternative interface, buttons on earphones or keyboard inputs could be used to stop/start or skip songs.

**Discussion:** One way to interact with web pages without using the screen is using a screen reader. A screen reader is part of the assistive technology (AT) that renders text and image content as speech or braille output. Users can use shortcuts on the keyboard to interact with the web pages. Popular screen readers include VoiceOver by Apple [45] and JAWS [58]. Such screen reader technologies have been developed for users who have visual impairments. Further research is necessary to understand the energy consumption of such assistive technology.

## 1.19 Avoid Extraneous Graphics and Animations

**Original:** "Graphics and animations are important for improving the user experience. However, they can also be battery intensive - use them with moderation [37]."

*Adaptation to Web.*

**Context:** Graphics and animations can be used anywhere on a GI. Since websites provide interactive GIs, this pattern can also be applied in web applications.

**Solution:** Observe the importance of graphics and animations on the user experience. Avoid using graphics animations or high-quality graphics. If necessary, resort to low frame rates and/or low resolution.

**Example:** The company has an example of a website, that uses simple, non-animated graphics to improve the user experience.

**Discussion:** Using graphics and animations only when they improve the user experience seems reasonable. For further explanations, also see Discussion in 1.16 Enough Resolution.

## 1.20 Manual Sync, On Demand

**Original:** "Perform tasks exclusively when requested by the user [37]."

*Adaptation to Web.*

**Context:** Some tasks, which are not strictly necessary for some application use cases, can be energy intensive.

**Solution:** Provide a mechanism that allows users to trigger energy-intensive tasks manually. For example, a button in the UI.

**Example:** The company perform actual calculations for a bill only when the user has requested it by clicking a button. They do not do operation-intensive pre-calculations on loading in case no bill was requested.

```
<template #button-bar>
<el-button
  type="primary"
  size="large"
  :loading="isBillingProcessRunning"
  @click="StartBillingCalculations"
>
  {{ $t('page.component.button.startBillingCalculations') }}
</el-button>
</template>
```

### Listing 11: Implementation of a button which starts the billing process only when pressed.

**Discussion:** Energy-intensive tasks (in the backend) do not influence the battery of the device that is visiting the website. One of the most frequently performed operations in the frontend is fetching data i.e., sending requests. These requests can indeed trigger energy-intensive tasks in the backend. Shneidermans eight golden rules of interface design state, that an application should support the (users) internal locus of control [8]. This means that users need to have control and freedom so that they can feel that they are in control of the system themselves.

## 2 RQ<sub>2</sub> INDUSTRY INSIGHTS ON ENERGY PATTERNS

We provide the guidelines in the supplement material. The guidelines are inferred from the questions asked to developers and their responses. We have also provided our coded answers We applied thematic analysis to developer excerpts to identify the rationale behind their concerns for energy patterns as shown in Figure 4 and Table 2.

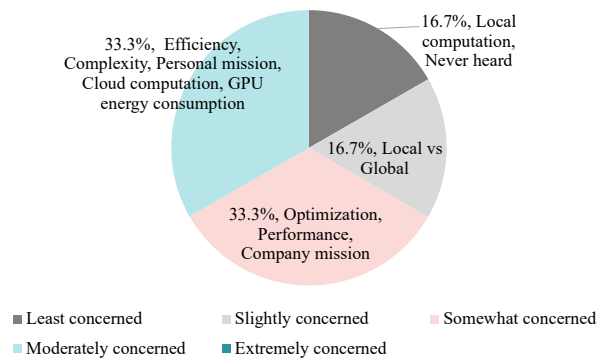


Figure 4: Developer concerns & motivations for patterns

## 3 RQ<sub>3</sub> IMPACT OF ENERGY PATTERNS

The supplement material contains the data that help simulate the two energy patterns, i.e., Dynamic Retry Delay and Open Only When Necessary.

## REFERENCES

- [1] 2023. Retrieved on Oct 2023 from <https://www.centurylink.com/home/help/internet/how-to-improve-gaming-latency.html#:~:text=The%20difference%20between%20high%20and%20low%20latency,->

**Table 2: Excerpts from Developers on their motivation for energy patterns and corresponding codes for the motivations**

Developers	Excerpts	Codes
P1 (moderately concerned)	<i>Energy Antipatterns correlates with efficiency and computational complexity.</i>	Efficiency, Complexity
P5 (somewhat concerned)	<i>It depends if its locally run, or on the cloud. If local, I don't really care. If GPUs are involved or the cloud, I am more concerned because I read about energy consumption of large data centers, and I know that GPUs consume a lot of power.</i>	Local computation, Cloud computation, GPU energy consumption
P6 (somewhat concerned)	<i>Application optimization and performance are still more important than energy optimization.</i>	Optimization, Performance
<p>Latency%20is%20the%20Generally%20an%20acceptable%20latency%20(or,a%20noticeable%20lag%20in%20gaming.</p> <p>[2] 2023. . Retrieved on Oct 2023 from <a href="https://sematext.com/blog/what-is-latency/#:~:text=In%20general%2C%20however%2C%20it's%20considered,load%20in%20under%203%20seconds">https://sematext.com/blog/what-is-latency/#:~:text=In%20general%2C%20however%2C%20it's%20considered,load%20in%20under%203%20seconds</a></p> <p>[3] 2023. . Retrieved on Aug 2023 from <a href="https://www.haivision.com/glossary/video-latency/">https://www.haivision.com/glossary/video-latency/</a></p> <p>[4] 2023. . Retrieved on Oct 2023 from <a href="https://www.oreilly.com/library/view/pattern-oriented-software-architecture/9780470845257/14_c004.html">https://www.oreilly.com/library/view/pattern-oriented-software-architecture/9780470845257/14_c004.html</a></p> <p>[5] 2023. . Retrieved on Oct 2023 from <a href="https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management?retiredLocale=de">https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management?retiredLocale=de</a></p> <p>[6] 2023. . Retrieved on Oct 2023 from <a href="https://www.iea.org/reports/data-centres-and-data-transmission-networks">https://www.iea.org/reports/data-centres-and-data-transmission-networks</a></p> <p>[7] 2023. <i>Android Authority</i>. Retrieved on Oct 2023 from <a href="https://www.androidauthority.com/ethernet-vs-wifi-1105145/">https://www.androidauthority.com/ethernet-vs-wifi-1105145/</a></p> <p>[8] 2023. <i>Capian</i>. Retrieved on Oct 2023 from <a href="https://capian.co/shneiderman-eight-golden-rules-interface-design">https://capian.co/shneiderman-eight-golden-rules-interface-design</a></p> <p>[9] 2023. <i>Digital Marketer</i>. Retrieved on Oct 2023 from <a href="https://www.digitalmarketer.com/blog/use-web-push-notifications/">https://www.digitalmarketer.com/blog/use-web-push-notifications/</a></p> <p>[10] 2023. <i>Dispose Pattern</i>. Retrieved on Oct 2023 from <a href="https://en.wikipedia.org/wiki/Dispose_pattern">https://en.wikipedia.org/wiki/Dispose_pattern</a></p> <p>[11] 2023. <i>Google Cloud</i>. Retrieved on Oct 2023 from <a href="https://cloud.google.com/scheduler/docs/reference/rest">https://cloud.google.com/scheduler/docs/reference/rest</a></p> <p>[12] 2023. <i>Greenspector</i>. Retrieved on Oct 2023 from <a href="https://greenspector.com/en/which-image-format-to-choose-to-reduce-its-energy-consumption-and-its-environmental-impact/">https://greenspector.com/en/which-image-format-to-choose-to-reduce-its-energy-consumption-and-its-environmental-impact/</a></p> <p>[13] 2023. <i>Intellipaat</i>. Retrieved on Oct 2023 from <a href="https://intellipaat.com/blog/best-web-development-languages/#1">https://intellipaat.com/blog/best-web-development-languages/#1</a></p> <p>[14] 2023. <i>Java EE</i>. Retrieved on Oct 2023 from <a href="https://javaee.github.io/tutorial/batch-processing001.html">https://javaee.github.io/tutorial/batch-processing001.html</a></p> <p>[15] 2023. <i>Mozilla</i>. Retrieved on Oct 2023 from <a href="https://developer.mozilla.org/en-US/docs/Web/API/Battery_Status_API">https://developer.mozilla.org/en-US/docs/Web/API/Battery_Status_API</a></p> <p>[16] 2023. <i>Mozilla</i>. Retrieved on Oct 2023 from <a href="https://developer.mozilla.org/en-US/docs/Web/WebDriver/Timeouts">https://developer.mozilla.org/en-US/docs/Web/WebDriver/Timeouts</a></p> <p>[17] 2023. <i>Opera</i>. Retrieved on Oct 2023 from <a href="https://www.opera.com/features/battery-saver">https://www.opera.com/features/battery-saver</a></p> <p>[18] 2023. <i>Python</i>. Retrieved on Oct 2023 from <a href="https://docs.python.org/3/library/logging.html#levels">https://docs.python.org/3/library/logging.html#levels</a></p> <p>[19] 2023. <i>Raider.io</i>. Retrieved on August 23 from <a href="https://raider.io/">https://raider.io/</a></p> <p>[20] 2023. <i>Statista</i>. Retrieved on Oct 2023 from <a href="https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/">https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/</a></p> <p>[21] 2023. <i>UDN</i>. Retrieved on Oct 2023 from <a href="https://udn.realityripple.com/docs/Web/API/Media_Streams_API">https://udn.realityripple.com/docs/Web/API/Media_Streams_API</a></p> <p>[22] 2023. <i>VueUse</i>. Retrieved on Oct 2023 from <a href="https://vueuse.org/core/useBattery/">https://vueuse.org/core/useBattery/</a></p> <p>[23] 2023. <i>Webkit</i>. Retrieved on Oct 2023 from <a href="https://webkit.org/">https://webkit.org/</a></p> <p>[24] Waleed Ali, Siti Mariyam Shamsuddin, and Abdul Samad Ismail. 2011. A Survey of Web Caching and Prefetching A Survey of Web Caching and Prefetching. <i>International Journal of Advances in Soft Computing and its Applications</i> 3 (03 2011).</p> <p>[25] Inmaculada Ayala, Mercedes Amor, Lidia Fuentes, and Michele Risi. 2019. An Energy Efficiency Study of Web-Based Communication in Android Phones. <i>Sci. Program</i>. 2019 (jan 2019), 19 pages. <a href="https://doi.org/10.1155/2019/8235458">https://doi.org/10.1155/2019/8235458</a></p> <p>[26] Wand B. 2023. <i>Weights &amp; Biases</i>. Retrieved on Oct 2023 from <a href="https://wandb.ai/site">https://wandb.ai/site</a></p> <p>[27] Aaron Bangor. 1999. Electronic Text Readability Issues for the Visually Impaired. <i>Proceedings of the Human Factors and Ergonomics Society Annual Meeting</i> 43, 23 (1999), 1372–1375. <a href="https://doi.org/10.1177/154193129904302323">https://doi.org/10.1177/154193129904302323</a></p> <p>[28] Kenneth C. Barr and Krste Asanović. 2006. Energy-Aware Lossless Data Compression. <i>ACM Trans. Comput. Syst.</i> 24, 3 (aug 2006), 250–291. <a href="https://doi.org/10.1145/1151690.1151692">https://doi.org/10.1145/1151690.1151692</a></p> <p>[29] John T. Bull. 2020. <i>GitHub:Vue-cam-demo</i>. Retrieved on August 23 from <a href="https://github.com/jbull328/vue-cam-demo/blob/master/src/components/CameraDisplay.vue">https://github.com/jbull328/vue-cam-demo/blob/master/src/components/CameraDisplay.vue</a></p> <p>[30] Aaron Carroll and Gernot Heiser. 2010. An Analysis of Power Consumption in a Smartphone. In <i>Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference</i> (Boston, MA) (USENIXATC'10). USENIX Association, USA, 21.</p> <p>[31] Mark Chang. 2023. <i>Chrome features</i>. Retrieved on Oct 2023 from <a href="https://blog.google/products/chrome/new-chrome-features-to-save-battery-and-make-browsing-smoother">https://blog.google/products/chrome/new-chrome-features-to-save-battery-and-make-browsing-smoother</a></p> <p>[32] C.-F. Chiasserini and E. Magli. 2002. Energy consumption and image quality in wireless video-surveillance networks. In <i>The 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications</i>, Vol. 5. 2357–2361 vol.5. <a href="https://doi.org/10.1109/PIMRC.2002.1046566">https://doi.org/10.1109/PIMRC.2002.1046566</a></p> <p>[33] Chiwoo Cho, Jinsuk Pak, Jinnyun Kim, Icksoo Lee, and Kijun Han. 2006. A Random Backoff Algorithm for Wireless Sensor Networks. In <i>Next Generation Teletraffic and Wired/Wireless Advanced Networking</i>, Yevgeni Koucheryavy, Jarmo Harju, and Villy B. Iversen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 108–117.</p> <p>[34] Shaiful Alam Chowdhury, Silvia Di Nardo, Abram Hindle, and Zhen Ming (Jack) Jiang. 2018. An exploratory study on assessing the energy impact of logging on Android applications. <i>Empirical Software Engineering</i> (10 July 2018), 1422–1456. <a href="https://doi.org/10.1007/s10664-017-9545-x">https://doi.org/10.1007/s10664-017-9545-x</a></p> <p>[35] Medium contributors. 2023. <i>GitHub: Reduce Size Example</i>. Retrieved on August 23 from <a href="https://medium.com/axiomzenteam/put-your-http-requests-on-a-diet-3e1e52333014">https://medium.com/axiomzenteam/put-your-http-requests-on-a-diet-3e1e52333014</a></p> <p>[36] MDN contributors. 2023. <i>Mozilla Page Visibility API</i>. Retrieved on Oct 2023 from <a href="https://developer.mozilla.org/en-US/docs/Web/API/Page_Visibility_API">https://developer.mozilla.org/en-US/docs/Web/API/Page_Visibility_API</a></p> <p>[37] Luis Cruz and Rui Abreu. 2019. Catalog of energy patterns for mobile applications. <i>Empirical Software Engineering</i> 24 (2019), 2209–2235.</p> <p>[38] Jim Daly, Peter Hecke, Divya Kamath, and Amol Deore. 2023. <i>Microsoft web API for Batch Processing</i>. Retrieved on Oct 2023 from <a href="https://learn.microsoft.com/en-us/power-apps/developer/data-platform/webapi/execute-batch-operations-using-web-api">https://learn.microsoft.com/en-us/power-apps/developer/data-platform/webapi/execute-batch-operations-using-web-api</a></p> <p>[39] Pranab Dash and Y. Charlie Hu. 2021. How Much Battery Does Dark Mode Save? An Accurate OLED Display Power Profiler for Modern Smartphones. In <i>Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services</i> (Virtual Event, Wisconsin) (MobiSys '21). Association for Computing Machinery, New York, NY, USA, 323–335. <a href="https://doi.org/10.1145/3458864.3467682">https://doi.org/10.1145/3458864.3467682</a></p> <p>[40] BooHoo Developers. 2022. <i>BooHoo</i>. Retrieved on Oct 2023 from <a href="https://www.boohoo.com/page/sustainability-guide.html">https://www.boohoo.com/page/sustainability-guide.html</a></p> <p>[41] Houssein Djirdeh, Addy Osmani, Mathias Bynens, and Pollard Barry. 2023. <i>Web.dev</i>. Retrieved on August 23 from <a href="https://web.dev/browser-level-image-lazy-loading/">https://web.dev/browser-level-image-lazy-loading/</a></p> <p>[42] Mian Dong, Yung-Seok Kevin Choi, and Lin Zhong. 2009. Power-Saving Color Transformation of Mobile Graphical User Interfaces on OLED-Based Displays. In <i>Proceedings of the 2009 ACM/IEEE International Symposium on Low Power Electronics and Design</i> (San Francisco, CA, USA) (ISLPED '09). Association for Computing Machinery, New York, NY, USA, 339–342. <a href="https://doi.org/10.1145/1594233.1594317">https://doi.org/10.1145/1594233.1594317</a></p> <p>[43] eventhelix. 2020. <i>Resource Manager Design Pattern</i>. Retrieved on Oct 2023 from <a href="https://www.eventhelix.com/design-patterns/resource-manager/">https://www.eventhelix.com/design-patterns/resource-manager/</a></p> <p>[44] Chris Harrelson. 2023. <i>batter-savings</i>. Retrieved on Oct 2023 from <a href="https://github.com/chrisht/battery-savings/blob/master/explainer.md">https://github.com/chrisht/battery-savings/blob/master/explainer.md</a></p> <p>[45] Apple Inc. 2022. <i>Accessibility - Vision</i>. <a href="https://www.apple.com/accessibility/vision/">https://www.apple.com/accessibility/vision/</a> Accessed: Oct 2023.</p>		

- [46] Tarunpreet Kaur and Dilip Kumar. 2016. TDMA-based MAC protocols for wireless sensor networks: A survey and comparative analysis. In *2016 5th International Conference on Wireless Networks and Embedded Systems (WECON)*. 1–6. <https://doi.org/10.1109/WECON.2016.7993426>
- [47] Stephan A. Kollmann and Alastair R. Beresford. 2017. The Cost of Push Notifications for Smartphones Using Tor Hidden Services. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroSPW)*. 76–85. <https://doi.org/10.1109/EuroSPW.2017.55>
- [48] Josephine Loo. 2023. *HTTP Request in Javascript*. Retrieved on Oct 2023 from <https://www.bannerbear.com/blog/5-ways-to-make-an-http-request-in-javascript/>
- [49] Rafal Mantiuk, Allan G. Rempel, and Wolfgang Heidrich. 2009. Display Considerations for Night and Low-Illumination Viewing. In *Proceedings of the 6th Symposium on Applied Perception in Graphics and Visualization* (Chania, Crete, Greece) (APGV '09). Association for Computing Machinery, New York, NY, USA, 53–58. <https://doi.org/10.1145/1620993.1621005>
- [50] Jan Monschke. 2019. *Garbage collection in source code*. Retrieved on Oct 2023 from <https://developers.soundcloud.com/blog/garbage-collection-in-redux-applications>
- [51] Jakob Nielsen. 1995. *Ten Usability Heuristics*. <https://www.nngroup.com/articles/ten-usability-heuristics/> Accessed: Oct 2023.
- [52] Zulie Pan, Yu Chen, Yuanchao Chen, Yi Shen, and Yang Li. 2022. LogInjector: Detecting Web Application Log Injection Vulnerabilities. *Applied Sciences* 12, 15 (2022). <https://doi.org/10.3390/app12157681>
- [53] Tanmay Patange. 2020. *Google Chrome's Battery-Savings*. Retrieved on Oct 2023 from <https://news.thewindowsclub.com/chrome-will-allow-websites-to-switch-to-battery-saving-mode-102329/>
- [54] Tomasz Peczek. 2019. *Demo.AspNetCore.PushNotifications*. Retrieved On Oct 2023 from <https://github.com/tpeczek/Demo.AspNetCore.PushNotifications/blob/main/Demo.AspNetCore.PushNotifications.Services.PushService/PushServicePushNotificationService.cs>
- [55] Julian Benadit Pernabas, Sagayraj Francis Fidele, and Krishna Kumar Vaithinathan. 2019. Enhancing Greedy Web Proxy caching using Weighted Random Indexing based Data Mining Classifier. *Egyptian Informatics Journal* 20, 2 (2019), 117–130. <https://doi.org/10.1016/j.eij.2019.01.001>
- [56] Cristian Popovici, Scala Steward, and Jason Webb. 2023. *Scala Retry*. Retrieved on Oct 2023 from <https://github.com/hipjim/scala-retry>
- [57] Allan Rempel, Rafal Mantiuk, and Wolfgang Heidrich. 2012. Display Considerations for Improved Night Vision Performance. *Final Program and Proceedings - IS and T/SID Color Imaging Conference* (05 2012).
- [58] Freedom Scientific. 2022. *JAWS Screen Reading Software*. <https://www.freedomscientific.com/Products/software/JAWS/> Accessed: Oct 2023.
- [59] Shopware Developer Team. 2023. *REST API Guide: Batch Mode Examples*. <https://developers.shopware.com/developers-guide/rest-api/examples/batch/> Accessed: Oct 2023.
- [60] Nuengwong Tuaycharoen, Veerawat Prodpran, and Boonsong Srithong. 2018. ClassSchedule: A web-based application for school class scheduling with real-time lazy loading. In *2018 5th International Conference on Business and Industrial Research (ICBIR)*. 210–214. <https://doi.org/10.1109/ICBIR.2018.8391194>