Degree Thesis, Åland University of Applied Sciences,
Bachelor of Information Technology/Bachelor of Engineering

DEVELOPMENT OF A TASK MANAGEMENT APPLICATION

- a Progressive Web Application

Kåre Hampf

2020:43

Publishing date: 20.01.2021
Supervisor: Björn-Erik Zetterman

DEGREE THESIS
Åland University of Applied Sciences

Study program:	Bachelor of Information Technology/Bachelor of Engineering
Author:	Kåre Hampf
Title:	Development of a Task Management Application
Academic Supervisor:	Björn-Erik Zetterman
Technical Supervisor:	

Abstract

The purpose of this thesis is to document the development of a cross-platform application for managing tasks stored in a specific plaintext formatted file residing on a server.

The application has been developed in JavaScript, HTML and CSS and uses technologies such as the Vue.js framework, WebDAV and IndexedDB.

The result is a JavaScript library conforming to the ES module standard with a Progressive Web Application (PWA) as a graphical user interface with functionality to manage tasks lists in a personal todo.txt formatted text file hosted on a Nextcloud server.

Keywords

JavaScript, ESM, Vue.js, PWA, todo.txt, WebDAV

Serial number:	ISSN:	Language:	Number of pages:
2020:43	1458-1531	English	36 pages

Handed in:	Date of presentation:	Approved on:
30.12.2020	17.12.2020	20.01.2021

EXAMENSARBETE
Högskolan på Åland

Utbildningsprogram:	Informationsteknik
Författare:	Kåre Hampf
Arbetets namn:	Utveckling av en applikation för hantering av arbetsuppgifter
Handledare:	Björn-Erik Zetterman
Uppdragsgivare:	

Abstrakt

Syftet med det här examensarbetet är att dokumentera utvecklingen av en plattformsoberoende applikation för att hantera uppgifter i en specifikt formaterad textfil på en server.

Utveckling av applikationen har skett i JavaScript, HTML och CSS och använder teknologier såsom ramverket Vue.js, WebDAV och IndexedDB.

Resultatet är ett JavaScript-bibliotek som följer ES-modulstandard med ett grafiskt användargränssnitt i form av en progressiv webbapplikation (PWA) med funktioner för att hantera uppgifter i en todo.txt-formaterad textfil på en Nextcloud-server.

Nyckelord (sökord)
JavaScript, ESM, Vue.js, PWA, todo.txt, WebDAV

Högskolans serienummer:	ISSN:	Språk:	Sidantal:
2020:43	1458-1531	Engelska	36 sidor

Inlämningsdatum:	Presentationsdatum:	Datum för godkännande:
30.12.2020	17.12.2020	20.01.2021

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Purpose

The purpose of this thesis is to document the development of a cross-platform application for managing tasks stored in a specific plaintext formatted file residing on a server.

From a user perspective, there are a few aspects to consider. The first is that data need to be resilient and that any changes in the task manager application must be synchronized. Another important aspect to consider is the possibility to use the task manager application on different platforms and allow for interaction with the data using other tools. These two aspects need to be tightly integrated so that a user with access rights can access information and update information from different platforms.

1.2. Method

A collection of common definitions, abbreviations and acronyms are included in the Appendix. Application development will be done using *cross-platform* software tools compatible with multiple operating systems such as *Microsoft Windows*, *Apple MacOS*, *GNU/Linux* and variants of *BSD*. Development will result in cross-platform software, also compatible with multiple platforms. The *JavaScript (JS)* programming language has been chosen for this project as it meets the above requirements.

Ideally the chosen *Integrated Development Environment (IDE)* should be multiple-language, supporting both *JavaScript (JS)* and the *PHP scripting language*. Meeting all requirements at least initially, *Code OSS* (the open source version of *Microsoft Visual Studio Code*) has been chosen as the IDE but other tools may be introduced at a later stage.

A source code repository hosted in the network using git[1] for version-control will provide incremental backups separate from the development environment. Publication under a suitable license can be made when the project has matured.

[1] https://git-scm.com/

1.3. Limitations

The separation into parts needs to be clear. A library for parsing and rendering should be reusable in other projects. A web application should be compatible with all major web browsers and a Nextcloud application should be implemented following good practices defined by the developers of Nextcloud. JavaScript will be run on clients while a Nextcloud application written in PHP runs on the server. Configuration of Nextcloud, WebDAV and backend server-side data storage mechanisms using file systems and/or SQL-databases will not be covered in this thesis. There are licensing issues when using external formats or relying on external libraries.

1.4. Background

The *todo.txt*[2] format defines a structure of tags and markers for describing tasks together with metadata and context using plaintext files. It was perhaps not originally invented by but named and popularized by Gina Trapani and an online community after a publication on the site Lifehacker[3] in 2006 (Trapani, n.d., 2006).

Plaintext is compatible with multiple, perhaps all platforms and can be manipulated using a vast number of text editors. Specialised tools for managing tasks in the todo.txt-format exist but not all provide user friendly graphical interfaces or offer cross-platform compatibility.

Nextcloud[4] is a free open source collaboration suite for hosting files similar to services such as *Dropbox* and *Google Drive*. Nextcloud also provides the infrastructure needed to run server-side applications but no dedicated tool for the todo.txt format exists in the Nextcloud ecosystem.

[2] http://todotxt.org
[3] https://lifehacker.com
[4] https://nextcloud.com/hub/

2. DESIGN AND TECHNOLOGIES

The implementation will consist of three main parts as illustrated by Figure 1:

1. A standalone, reusable, **library** written in JavaScript
2. A **Progressive Web Application**[5] written in JavaScript with a *User Interface (UI)* using HTML and styled with CSS
3. A **Nextcloud[6] application** written in the PHP scripting language

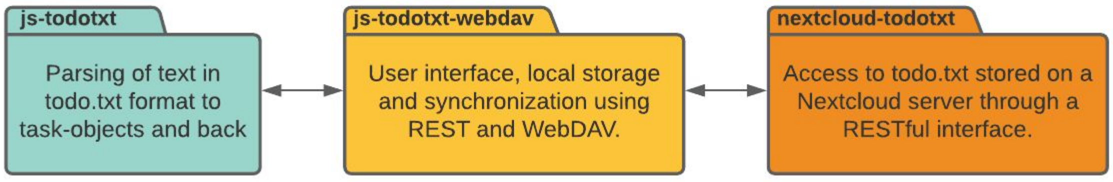

js-todotxt
Parsing of text in todo.txt format to task-objects and back
js-todotxt-webdav
User interface, local storage and synchronization using REST and WebDAV.
nextcloud-todotxt
Access to todo.txt stored on a Nextcloud server through a RESTful interface.

Figure 1. Illustration of the project divided into parts.

The JavaScript library will be compatible with lists of tasks stored in plaintext conforming to the todo.txt format. The library will include functionality to parse lists to an internal format and back to plaintext for synchronization purposes.

The *Progressive Web Application (PWA)* provides the user with a graphical user interface for manipulating tasks using the JavaScript library. Functionality in PWAs are built using JavaScript and the structure of the graphical user interface is defined in HTML with layout and styling in CSS (*Web Design and Applications - W3C*, n.d.).

The Nextcloud server acts as a host and serves plaintext files and the PWA which enables the user to load the web application inside a Nextcloud application (making it a hybrid PWA). The web application should ideally also as a pure PWA be able to interact directly with the

[5] https://web.dev/what-are-pwas/
[6] https://nextcloud.com/

WebDAV *API* provided by the Nextcloud or any other WebDAV[7] server. Both scenarios are illustrated in Figure 2.

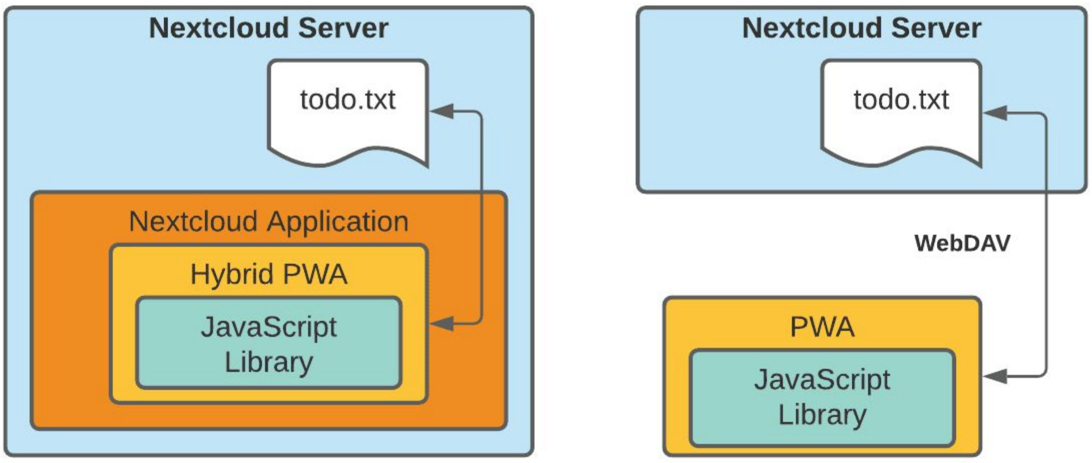


Figure 2. Illustration of a hybrid PWA Nextcloud application alongside a pure PWA implementation.

2.1. Cross-platform compatibility

Development tools used and the software built using them should ideally both be *cross-platform software* and *platform-independent*. Cross-platform applications may run on different operating systems and hardware platforms (Wikipedia contributors, 2020e).

Web browsers are used to access information on the *World Wide Web* from a wide variety of existing devices. It is estimated that around 5 billion people use a browser globally and the most popular browsers, *Google Chrome*[8] and *Mozilla Firefox*[9] together with their derivatives (e.g., Chromium, Microsoft Edge) are cross-platform software compatible with all major operating systems (Wikipedia contributors, 2020d).

[7] https://www.rfc-editor.org/info/rfc4918
[8] https://www.google.com/chrome/
[9] https://www.mozilla.org/en-US/firefox/browsers/

2.2. Design patterns

For libraries where reusability and an extendable interface to build upon are desired the *module* or *revealing module* pattern is a good choice. Modules are contained to their own scope which limits pollution of the global namespace (Paltoglou et al., 2018, sec. III a).

The graphical task manager application will use a pattern resembling MVP (Model-View-Presenter) as the view in the browser or in the interface provided by the browser already acts as a passive view, rendering the output in a presenter using templates and from HTML generated in a model containing the business logic.

2.3. JavaScript and ECMAScript

Applications for mobile devices and desktops have traditionally been implemented by compiling source code into binary executables. The compatibility of binary executables are limited to a single platform and a limited number of devices or operating systems (Schneider & Gersting, 2018, p. 495).

JavaScript is a scripting language conforming to the *ECMAScript (ES)* specification standardised by the *European Computer Manufacturers Association (ECMA)*, renamed as *Ecma international* in 1994 (Wikipedia contributors, 2020a). Scripting languages are not run from binary executables but instead interpreted at runtime inside a host environment. JavaScript was at first designed to be a scripting language for the web providing functionality for elements defined inside the markup language of web pages, for instance, the functionality of a button element in a form using the *onclick attribute*. Over time JavaScript has evolved into a general purpose language with a wide range of uses. While still commonly found running inside web browsers, today JavaScript also runs outside of web browsers on desktops, servers, smartphones and embedded devices. The *Node.js* runtime environment runs JavaScript on servers without the need for graphical environments and *Espruino* is an example of an interpreter for running JavaScript on microcontrollers (Williams, 2017). "These days, it's safe to say that JavaScript is everywhere" (Stefanov & Sharma, 2013, Chapter 1).

2.4. Object-Oriented Programming

Defining objects containing data and code and the encapsulation of objects in classes with inheritance using *Object-Oriented Programming (OOP)* in high-level languages originates in the 1950s (Wikipedia contributors, 2020b).

The concept of classes were introduced in ECMAScript 2015 as *syntactic sugar*, alternate syntax for already existing features. Classes in JavaScript are special functions subject to stricter syntax with the ability to provide a *constructor* function with access to functions inherited from a superclass (Mozilla and individual contributors, 2020a). In JavaScript properties are always public so encapsulation of classes will in contrast to other OOP languages not provide information hiding into categories such as public, private or protected (Stefanov & Sharma, 2013, p. 17).

Key concepts in OOP (Stefanov & Sharma, 2013, p. 15):

- Objects with attributes and methods (properties and functions)
- Classes as blueprints for objects
- Encapsulation of data with methods for manipulating the data using abstraction
- Aggregation of multiple objects into more complex objects
- Inheritance of objects allowing for reusability with modification
- Polymorphism where different objects can share common attributes and methods

The separation of JavaScript namespaces into modules can be performed using OOP with polymorphic classes and using inheritance. Implementing module functionality using principles of OOP design throughout seems ideal when it comes to designing reusable libraries.

2.5. Progressive Web Application (PWA)

The term Progressive Web Application was coined by designer Frances Berriman and Google Chrome engineer Alex Russel in 2015 (Russell, 2015; Wikipedia contributors, 2020c). PWAs are built using a combination of existing technologies like HTML, CSS and JavaScript. Using technologies that already exist in billions of devices connected to the internet PWAs aim to give users a platform native experience similar to what they are already used to. The experience of a PWA is predominantly dependent on the application being fast, reliable and engaging (Sheppard, 2017, p. 6). See Figure 3 for the "community-approved" logo representing the concept of PWAs (Salnikov, 2017).

Figure 3. Logo for Progressive Web Applications by Diego González-Zúñiga (CC0).

What makes PWAs "progressive" is that they use emerging web browser features such as secure contexts (use HTTP with SSL encryption), are described in a *application manifest* and include *service workers*[10] that can perform tasks in the background (Mozilla and individual contributors, 2020c). PWAs use smart caching where the application can present the user with cached data immediately on load and refresh the displayed contents after successful synchronization. They can be described as standalone web pages that run their own code with the ability to access data independently of their web server host and are compatible with all platforms that provide a standards compliant web browser. Additionally PWAs may use mechanisms for updating their own running code when a newer version is available and they can also receive *push notifications*[11] from their host. To make PWAs look less like web browser windows and more like native applications they can display a splash screen while loading.

[10] https://web.dev/service-worker-mindset/
[11] https://web.dev/push-notifications-overview/

2.6. Client-side data storage

Persistent data in JavaScript running in web browsers has traditionally been stored using *JavaScript Object Notation (JSON)* and the *Window.localStorage API*. This API is not available to workers and service workers in a PWA running in an offline state (Sheppard, 2017, Chapter 5).

A complex transactional database named *IndexedDB*[12] capable of storing multiple types of data is available to PWAs. *Wrappers* that simplifies its use exist, for example, *Dexie*[13] by David Fahlander. While some wrappers try to simplify the API, others provide relational SQL-like syntax (e.g., *JsStore*[14] by Ujjwal Gupta, *Lovefield*[15] from Google) so again a choice to be made.

A wrapper library named *localForage* created by the Mozilla team also provides a fallback to localStorage if the client platform should lack support for IndexedDB (Mozilla and individual contributors, 2013). For this project localStorage would in principle be sufficient (but being synchronous and consequently lock up the application while loading data) so localForage was chosen.

2.7. Nextcloud application

The Nextcloud documentation includes an example that was used as a base for building a custom application. The example includes a custom API using *Representational State Transfer (REST)* and a frontend user interface built using the Vue.js[16] framework that consumes the API (Nextcloud GmbH, 2020).

[12] https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API
[13] https://dexie.org/
[14] https://jsstore.net/
[15] https://google.github.io/lovefield/
[16] https://vuejs.org/

3. APPLICATION DEVELOPMENT

3.1. Preparation

Before writing any code some decisions were made and a couple experiments carried out. First the ISC[17] software license was chosen for its suitability and widespread use in similar applications. Multiple environments, utilities and frameworks were evaluated before being used in the project.

3.1.1. Integrated Development Environment

The desire for cross-platform compatibility did not limit the choices as many modern IDEs are platform independent. Examples are *Eclipse*[18], *Apache Netbeans*[19], *Microsoft Visual Studio (VS) Code*[20], *Jetbrains WebStorm*[21] *and PHPStorm*[22].

The todo.txt JavaScript library was developed using a text editor (vim/gvim[23]) in combination with VS-Code. This environment was then also used to create a simple PWA and a prototype version of the todo.txt application. All dynamic elements of the output were created programmatically using template strings. Plugins and add-ons to VS-Code could handle small parts of the work but this was a slow process.

A second prototype of the task manager application was created in the WebStorm IDE by importing existing the prototype codebase but this provided only modest improvements. The IDE made the process of splitting up the codebase into smaller parts easier as WebStorm could analyze sources. At this point the decision was made to start over using a template and then incorporate parts of the existing code instead. The task manager application was then fully created in PHPStorm (an almost identical IDE to WebStorm but with additional support for the PHP scripting language that would later be needed for the Nextcloud application).

[17] https://opensource.org/licenses/ISC
[18] https://www.eclipse.org/
[19] https://netbeans.org/
[20] https://code.visualstudio.com/
[21] https://www.jetbrains.com/webstorm/
[22] https://www.jetbrains.com/phpstorm/
[23] https://www.vim.org/about.php

3.1.2. ECMAScript Modules

When the source code grew in size a desire to split it into manageable parts led to confusion over module standards. Node.js was the first JavaScript interpreter to support module loading implementations but it has multiple standards. One of the module standards is *CommonJS* which uses a *require* statement to load dependencies from external sources. As the standards in use by Node.js are incompatible with JavaScript in web browsers tools for *bundling* external dependencies into standalone and browser-compatible JavaScript files were developed. Bundlers like *Browserify* and *Webpack* offer a method to incorporate modules in *transpiled* (translated and "compiled"), backward and web browser compatible form (Webpack contributors, n.d.).

A new standard for JavaScript modules included in the 6th edition of the ECMAScript specification, ECMA-262 (more commonly known as ES6 or ECMAScript 2015), adds support for *ECMAScript-modules* (ES-modules, or simply *ESM*). This standard is endorsed by the W3C and support for it has become widely available at least in web browsers (and also in Node.js with some help from third party libraries).

3.1.3. Proof of concept PWA

The original plan was not to use any frameworks for designing a PWA and a "proof of concept" PWA was created as a sort of a warm-up. The idea was to implement a simple library providing algorithms (a generator of sequences of turns needed to scramble a Rubik's cube), perform unit testing and build a simple PWA. This side project was named *Cubescram* and was created based on an example accompanied by a tutorial named *Hello-PWA* by James Johnson (Johnson, 2018). The library was a simple browser compatible ES module with a manually crafted HTML generator and unit testing was successfully performed using the JavaScript unit testing framework *QUnit*[24].

3.1.4. Transpiling JavaScript and modules into bundles

After initial tests it became apparent that browser compatibility was not yet as widespread as hoped for. While most browsers supported PWAs and also could load ES modules, the latter

[24] https://qunitjs.com/about/

gave reason for looking more into *transpiling* JavaScript modules into backward compatible *bundles* with better compatibility with JavaScript interpreters that are limited to older standards. Bundlers additionally offer functionality to include assets like images, fonts and CSS in a more manageable way (Webpack contributors, n.d.). The task manager application prototype was built using *Webpack* (see Figure 4).

Figure 4. Webpack logo use permitted by the JS Foundation Trademark Policy.

After having used Webpack for the first prototype of the task manager application another bundler, *Rollup* (see Figure 5), was found to be better suited for bundling the JavaScript parser library into a Node.js module package as it uses a *tree-shaking* technique to exclude unused parts of other imported libraries making resulting output smaller (Rollup contributors, 2020).

Figure 5. Rollup logo by Julian Lloyd released under the MIT license.

3.1.5. JavaScript utilities and frameworks

3.1.5.1. Lodash

There are a number of built-in methods in JavaScript for manipulating arrays and lists but additional methods from the utility library *Lodash* (see Figure 6) proved useful. Lodash utilities are compatible with both Node.js and web browsers, loadable from ES modules and provide backward compatibility with older versions of JavaScript (Lodash team and contributors, 2009).

Figure 6. Lodash logo released under the MIT license.

3.1.5.2. Vue.js

The need for a separate component in the user interface for altering settings and the thought of having a small status display included somewhere lead to the realisation that a single page design might not be the best choice. Up until this point the view was nothing more than a list with some buttons on top.

The example Nextcloud application tutorial uses the framework *Vue.js* (see Figure 7) and it is aimed at being lightweight and incrementally adoptable. This meant it could be conveniently incorporated in the task manager application and provide a better user interface. Using Vue.js for the user view would still be using JavaScript and HTML but with separation of user interface parts into components and allowing for communication between components using properties and events (Gerchev, 2018).

Figure 7. Vue.js logo by Ethan You (CC BY 4.0).

In a web browser the contents are represented internally as a HTML *Document Object Model (DOM)*. Vue.js keeps a virtual DOM representation that makes components written using templates able communicate using efficient two-way data bindings. A component on a web page can for instance display the contents of a variable and the displayed value will update automatically when the value of the variable changes (Rojas, 2019, Chapter 5).

3.1.6. Project directory structure

A main directory with subfolders containing all parts, each in turn divided into subfolders according to best practices of that programming language.

3.1.6.1. JavaScript library directories and file naming conventions

ECMAScript modules should ideally use a *.mjs* file extension and be included in other projects using the *import* statement. A common practice is to put the source code of the entry point JavaScript file *index.js* into a *src*-folder and external files in suitable subfolders of that directory. Bundlers then process the src-folder and generate output in another specified folder.

3.1.6.2. PWA directories

In addition to directories used by libraries, applications might also have assets such as fonts, icons and images that need to be placed in appropriate folders. Bundlers will then wrap assets up in bundles and include them in the output.

3.1.7. Code repository setup

The initial thought was to upload everything to GitHub. Keeping all the source code public and out in the open before the task management application was usable and yet not stable, likely to undergo major changes, seemed after consideration less of a good idea. Initial testing might also involve hardcoding of credentials not meant to be shared publicly.

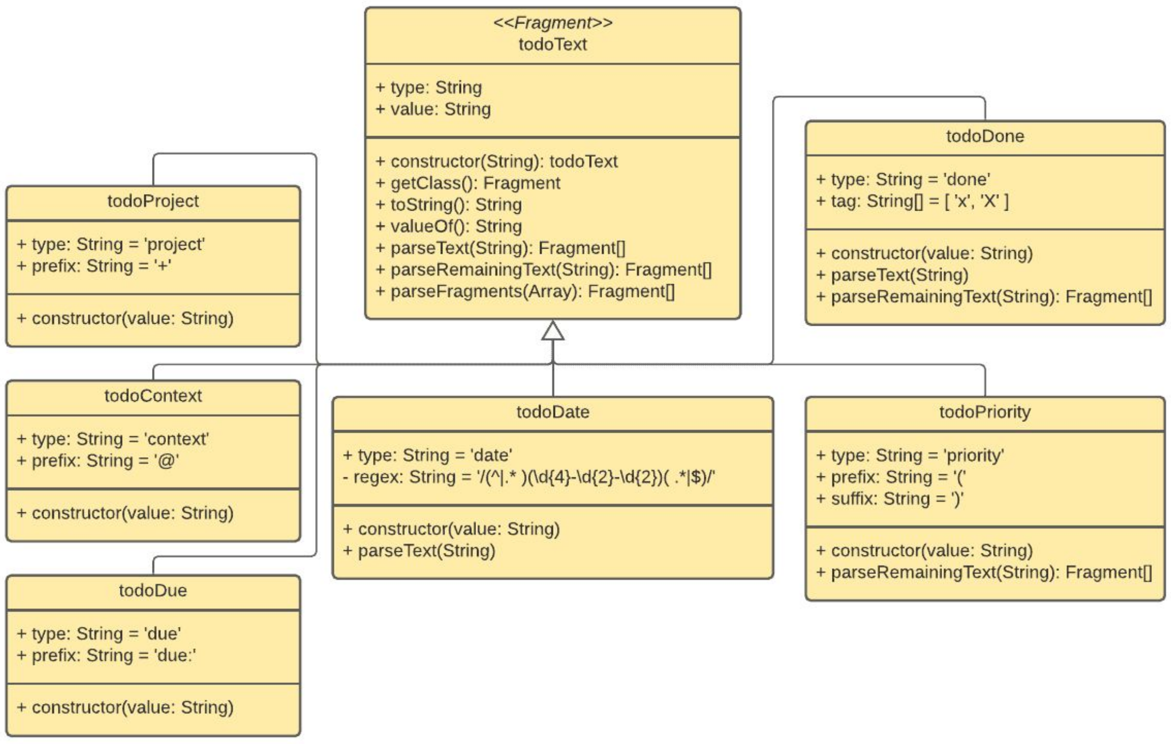
A software repository was required both for version control and for keeping backups separate from the development workstation. Instead of using a public repository server, a private server in the local network was deemed sufficient. A minimal, in comparison with other options less resource-hungry, alternative named *gitolite*[25] worked out great.

3.2. Development

3.2.1. JavaScript library for parsing the todo.txt format

The JavaScript *todo.txt* format parsing library was written as multiple ES modules with classes for supported markers in todo.txt data. Some classes have specific parser functions (for example due-dates) while other classes with similar markers use a common inherited prefix and/or suffix processor. The library creates an array consisting of all lines in the input and recursively processes them using the individual marker-specific parser classes as shown in Figure 8.

[25] https://gitolite.com/gitolite/



<<Fragment>>
todoText
+ type: String
+ value: String
+ constructor(String): todoText
+ getClass(): Fragment
+ toString(): String
+ valueOf(): String
+ parseText(String): Fragment[]
+ parseRemainingText(String): Fragment[]
+ parseFragments(Array): Fragment[]
todoProject
+ type: String = 'project'
+ prefix: String = '+'
+ constructor(value: String)
todoDone
+ type: String = 'done'
+ tag: String[] = ['x', 'X']
+ constructor(value: String)
+ parseText(String)
+ parseRemainingText(String): Fragment[]
todoContext
+ type: String = 'context'
+ prefix: String = '@'
+ constructor(value: String)
todoDate
+ type: String = 'date'
- regex: String = '/(^|.*)(\d{4}-\d{2}-\d{2})(.*|$)/'
+ constructor(value: String)
+ parseText(String)
todoPriority
+ type: String = 'priority'
+ prefix: String = '('
+ suffix: String = ')'
+ constructor(value: String)
+ parseRemainingText(String): Fragment[]
todoDue
+ type: String = 'due'
+ prefix: String = 'due:'
+ constructor(value: String)

Figure 8. Class diagram of todo.txt element submodules.

After development and testing of the library yielded a usable result *Rollup* was used to create several JavaScript bundles compatible with multiple targets (*ESM*, *CommonJS* and *Universal Module Definition* formats). The bundles were then compressed into a distributable archive.

3.2.1.1. Serializing todo.txt into JSON

Representing data as attributes with values for use in communication or storage can be done by using several different standardised formats. Common formats are *Extensible Markup Language (XML)* and the open and in comparison somewhat relaxed standard *JavaScript Object Notation (JSON)* (ECMA, 2017). Not exclusively used by JavaScript (although the syntax resembles JavaScript and is the origin of its name), JSON is considered the standard format for web applications (Rischpater, 2015, Chapter 1).

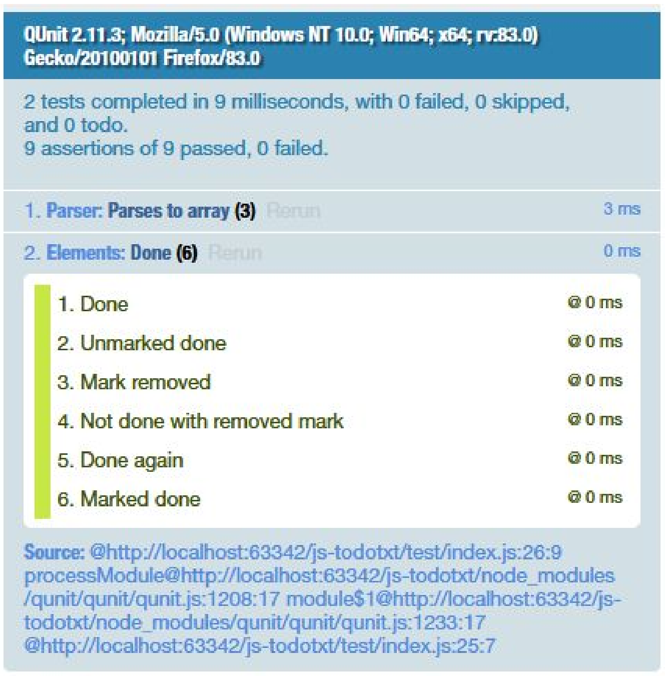
Using a modular design implies components or parts that need to communicate data between the graphical JavaScript frontend, the data model and the server PHP backend. Using JSON as a standardised format for representing data makes this a standardized procedure.

Parsing the task list from plaintext generates an array of classes containing both data and code. Arrays can in turn be serialized to text using JSON with the advantage that this process “washes” the classes clean from code (functions) but leaves the attributes and properties generated by the parsing process. Having parsed output in JSON also allows for easy storage of data, conversion to HTML for displaying on a web page and conversion back into plaintext for saving and synchronization.

3.2.1.2. Testing of the JavaScript library

A number of unit tests were written for the element parsers and the js-todotxt library in early stages of development. Writing tests compatible with both web browsers and Node.js proved difficult but the tests were ultimately compatible with both environments. This provided the means to run tests with output both in terminal environments and in web browsers.

The element unit tests parse todo.txt formatted text to task objects and then compares them to expected output in serialized JSON form. This is then parsed back to plain todo.txt and compared to the original input. Utility functions were tested by verifying returned values by comparing them to predefined values. Example output from QUnit is shown in Figure 9. Unit testing simplified the development of the JavaScript library to a significant extent.


```
QUnit 2.11.3; Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0)
Gecko/20100101 Firefox/83.0

2 tests completed in 9 milliseconds, with 0 failed, 0 skipped,
and 0 todo.
9 assertions of 9 passed, 0 failed.

1. Parser: Parses to array (3) Rerun                          3 ms
2. Elements: Done (6) Rerun                                   0 ms
   1. Done                                                  @ 0 ms
   2. Unmarked done                                         @ 0 ms
   3. Mark removed                                          @ 0 ms
   4. Not done with removed mark                            @ 0 ms
   5. Done again                                            @ 0 ms
   6. Marked done                                           @ 0 ms

Source: @http://localhost:63342/js-todotxt/test/index.js:26:9
processModule@http://localhost:63342/js-todotxt/node_modules
/qunit/qunit/qunit.js:1208:17 module$1@http://localhost:63342/js-
todotxt/node_modules/qunit/qunit/qunit.js:1233:17
@http://localhost:63342/js-todotxt/test/index.js:25:7
```

Figure 9. Screenshot of unit testing using QUnit on a todo.txt element parser.

3.2.2. JavaScript PWA

3.2.2.1. Reading of text files

At first a single web page was written in HTML, containing a button element which could bring up the file chooser and let the user upload a file from the local filesystem. The application then listed the parsed contents using functions for generating HTML. While developing the JavaScript library the upload button-approach was sufficient but having to constantly upload files while testing became a chore. Automatically accessing the local filesystem outside the sandboxed JavaScript environment has been proposed as a standard Web API (Mozilla and individual contributors, 2020b), but JavaScript in browsers can currently not read from the local filesystem without using non-standard browser-specific APIs or external components written in *ActiveX* or *Java*.

3.2.2.2. Retrieval of data using the Nextcloud API

The todo list could be displayed using functions for generating HTML inside a Nextcloud application which was based on a tutorial application included in the Nextcloud documentation (Nextcloud GmbH, 2020). In this environment it was natural to use GET and PUT methods from the Nextcloud API for reading and writing plaintext files which also lead to a dependency on Nextcloud.

3.2.2.3. Retrieval of data using WebDAV

Pages on the *World Wide Web* were in the beginning read-only files in hypertext format hosted by servers and retrieved by using the *Hyper-Text Transfer Protocol (HTTP)*. Making these files available on a server required copying them from some sort of removable storage media or uploading them remotely over some other protocol such as the *File Transfer Protocol (FTP)*. To extend the functionality of HTTP, the *World Wide Web Distributed Authoring and Versioning (WebDAV)*, enabled remote authenticated clients to upload files (IETF Group, IETF Network & Others, 2007).

Ideally the task managing application should also work fully outside of the Nextcloud server or perhaps even without using Nextcloud altogether with the file stored on some different kind of server. Nextcloud does provide a standards compliant WebDAV service and if the application were to use this it would be compatible with but not dependent on Nextcloud. Support for transferring the todo.txt file using WebDAV presented another challenge as long as the application was not directly hosted using the same web server, overcoming *Cross-origin Resource Sharing (CORS)* issues. If JavaScript requests resources from a source that is different than its own origin, the browser will query the host of the resource whether this is a valid request or not. The conclusion is that the server needs to be configured to allow requests from scripts loaded from other origins (Mozilla and individual contributors, 2019).

When the web application is hosted on the same web server as the todo.txt file they share a common origin but in all other scenarios CORS needs to be configured to allow requests. While Nextcloud provides WebDAV server it cares less about CORS. After failed attempts at using a number of Nextcloud helper applications that were supposed to be able to configure CORS inside the Nextcloud server, the solution was ultimately to reconfigure the hosting Apache web server and then manipulate HTTP headers outside of Nextcloud. This allowed for development of the PWA independently of Nextcloud while still maintaining compatibility with its WebDAV implementation.

3.2.2.4. Icon design

Nextcloud applications use a 32 by 32 pixels icon and support vector graphics. Icons in *Scalable Vector Graphics (SVG)* format were created using the *Inkscape*[26] open source vector graphics editor and can be seen in Figure 10. These icons can also be used for example in splash screens.

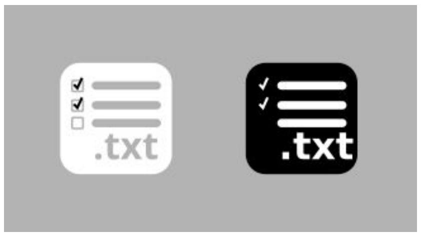


Figure 10. Samples of the icon design.

[26] https://inkscape.org/

Several tools for generating bitmap versions of the icons in all needed sizes can be found as Node.js *npm*-packages together with tools that generate package manifests for PWAs.

3.2.2.5. Responsive styling using CSS

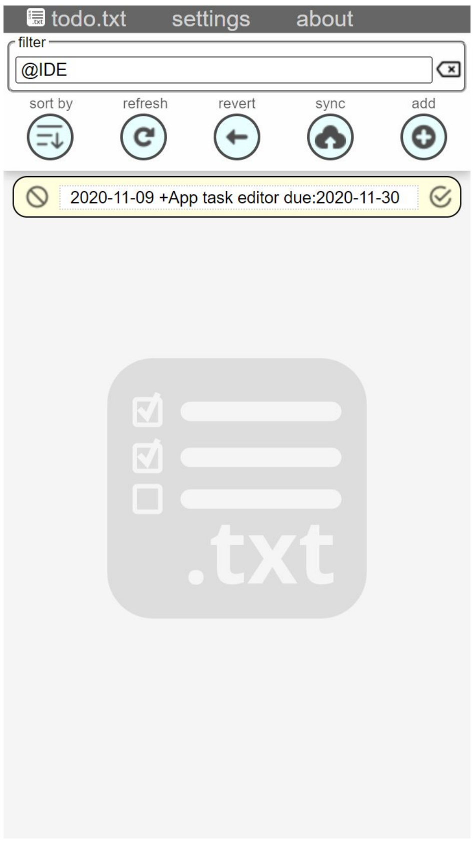
Stylesheets in CSS format were created throughout the development of the application. It became a continuous part of the development process to style the user interface of the task manager application, in part to aid in the spotting of errors in generated HTML pages and CSS.

3.2.2.6. Persistence of data inside the browser

Credentials for accessing the todo.txt file on the server are stored inside the client web browser using IndexedDB through the localForage wrapper (Mozilla and individual contributors, 2013). To allow the application to work in a disconnected state while being offline, a copy of the remote todo.txt data is also cached on download allowing tasks to be manipulated and then synchronized at a later stage when the user is back online.

3.2.2.7. Task editor

A text input element styled to look like an existing task, as seen in Figure 11, accepts the input of new tasks and inserts them in the local data store for synchronization after parsing. When the user wants to edit an existing task the task is converted to plaintext and preloaded in the editor. A date picker and other interactive elements could be added, for instance, a dropdown suggesting existing project- and content-tags.



@IDE

2020-11-09 +App task editor due:2020-11-30

Figure 11. Editing of a task in the todo.txt application.

3.2.2.8. Vue.js components

The need for a separate settings dialogue in the task manager application made it clear that developing the application without using any framework was a needlessly difficult process. Creating forms on additional web pages is a minor task but managing the links pointing from one page to another is likely to break functionality at some point.

Routing between screens in a web application turned out to be easily accomplished in *Vue.js* using components. Pages can themselves consist of multiple components that appear dynamically. When creating new tasks or editing existing tasks in the application all code and stylesheet matter belonging to the task editor could be placed inside a separate and reusable Vue.js component template and then included where needed (Rojas, 2019, Chapter 5).

The framework Vue.js is extendable by the use of plugins that, for example, provide support for managing PWA manifests and service workers. Internally Vue.js uses the exact same libraries that were planned to be used in the task manager application in the first place, for instance *Google Workbox* for service workers and *Webpack* for bundling and generating distributable archives. Additionally Vue.js offered a layer of abstraction on top of those libraries with the option to choose between multiple similar solutions without major changes to existing code, still with the ability to manually adjust the configurations of those underlying libraries if necessary.

3.2.2.9. Listing tasks using Sortable.js and Vue.Draggable

A desire to make the task list interactive in combination with adding functionality by loading third-party modules resulted in trying out *Sortable.js*[27], a JavaScript library providing functionality for making list items in web browsers manually sortable. When the task manager application started to use the Vue.js framework a Vue component named *Vue.Draggable*[28] (see Figure 12) was found that provided the features of Sortable.js and allowed for integration with the view model in Vue.js.

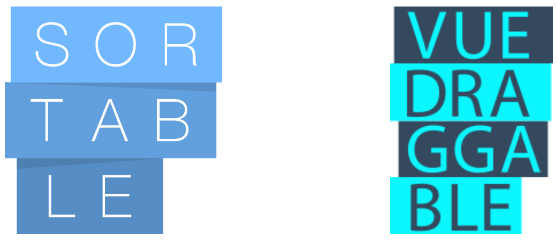


Figure 12. Sortable.js and Vue.Draggable logos (both projects released under the MIT license).

3.2.2.10. Task sorting and filtering

Sorting of the displayed task list was initially implemented using simple buttons in the user interface and sorting tasks by the value of different tags and attributes was accomplished using the object-oriented design of the JavaScript todo.txt library where all elements had type attributes and comparable values. At early stages of development sorting was only performed in the view and not stored as state. When the task list in the user interface was reimplemented as a Vue.js component the handling of the tasks in an internal array made dynamic updates of changes in the displayed list possible. Tasks could now be manually sorted by dragging them

[27] https://sortablejs.github.io/sortablejs/
[28] https://github.com/SortableJS/Vue.Draggable

around as well as automatically reordered using sorting methods called from the user interface.

Filtering was implemented as a text input element that matched words against existing tasks and a mechanism in HTML provided means to add a suggestions list containing all existing tags like project names and contexts for easy insertion as filter input.

3.2.2.11. Installation functionality on desktops and smartphones

Hosting the task manager application on a web server makes it compatible with most web browsers and operating systems. Even browsers compatible only with older standards can run the application, albeit not as well as most of the libraries in use fall back to older, often slower methods. On browsers with explicit support for PWAs a *pop-up* will indicate that the application is installable and that it can be added to the home screen or added to the start menu. This has been tested in *Windows 10*, *Linux* and on *Android* smartphones with good results.

3.2.3. Nextcloud application in PHP

3.2.3.1. Interaction between the Nextcloud application and the PWA

If the user is logged in to the Nextcloud server, a PWA hosted from a Nextcloud application can use the Nextcloud API and access the todo.txt file directly, or the Nextcloud application could provide custom REST interfaces. Otherwise the client needs to authenticate using Nextclouds login-flow which is trivial to do in a completely separate application like for example in a native Android application but difficult when using JavaScript running inside a web browser due to Cross-Origin Resource Sharing (CORS) issues.

3.2.4. Deployment and publication

The task manager application was deployed on a web server and hosted online using proper SSL certificates as illustrated by Figure 13. The application is not dependent on any specific web server. Any standard web server would work, including solutions such as static hosting from cloud storage. To allow for users to test the application they simply need to be given the web address to visit using their web browsers. The todo.txt file hosting service setup is specific to the chosen WebDAV server (in this case Nextcloud) and comes with its own requirements. Additional development, testing and cleanup was needed before the codebase and the task manager application could be publicly distributed.

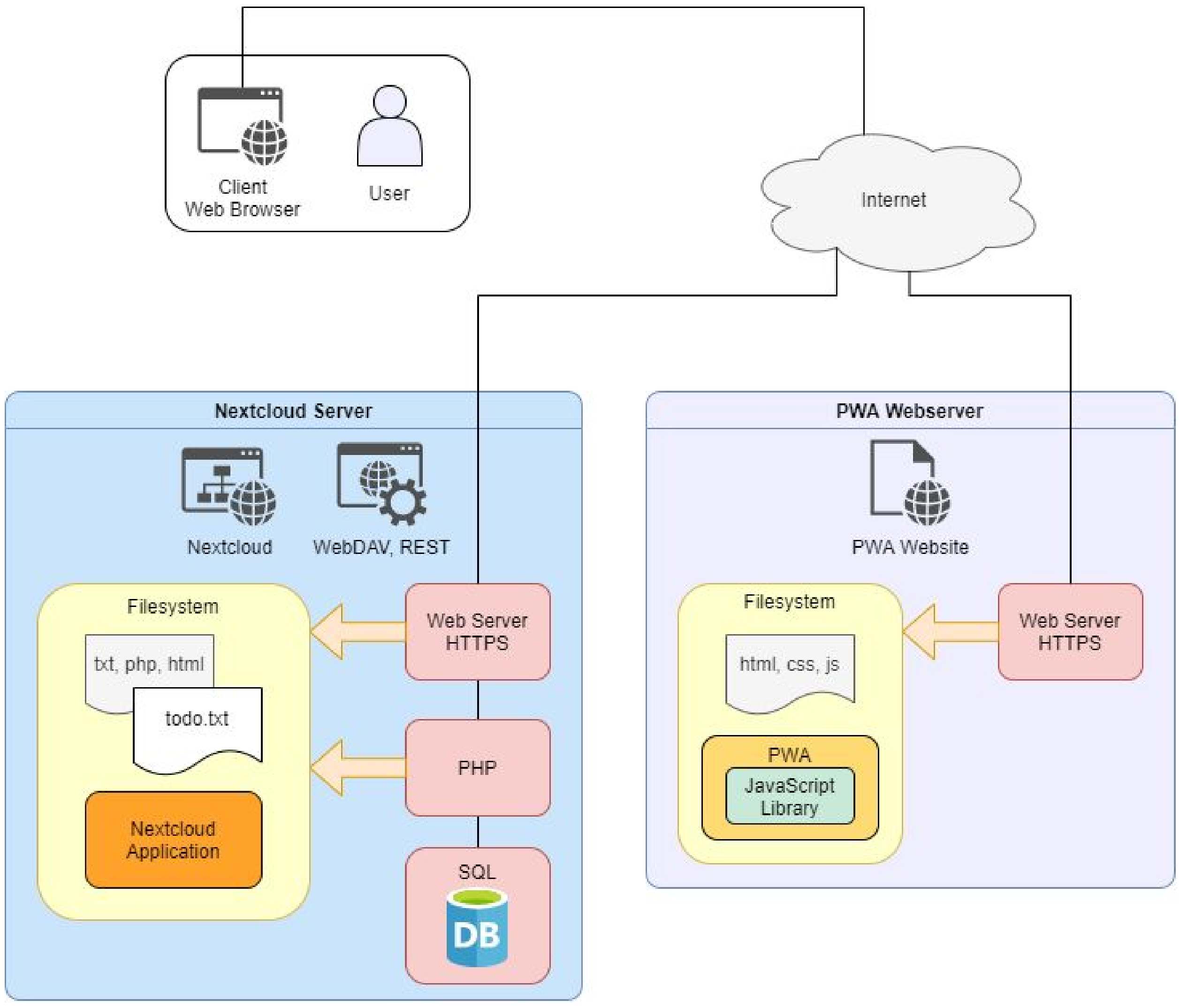


Figure 13. Deployment illustration.

4. CONCLUSION

4.1. Result

4.1.1. The js-todotxt library

The JavaScript library js-todotxt is converted into other versions for compatibility, or *transpiled*, into a number of bundles that then can be imported into other projects.

4.1.1.1. Installation of the library

Before possible publication in a JavaScript package repository the library can be manually downloaded as a compressed archive. It can then be installed as a Node.js module dependency (i.e. added to the *package.json* file) using a Node.js package manager as shown in Figure 14.

```
$ npm install todotxt-v1.0.9.tgz
```

Figure 14. Example of installing the js-todotxt library into a project using shell command.

4.1.1.2. Using the library

The library can be imported as an ES module using the *import statement* as shown in Figure 15 (alternatively by *CommonJS require-statement*) after which the methods provided by the library are available. A selection of the available methods are listed in Table 1.

```
import TodoTxt from 'todotxt';
let taskJSON = TodoTxt.getJSONFromTodoTxt(text);
```

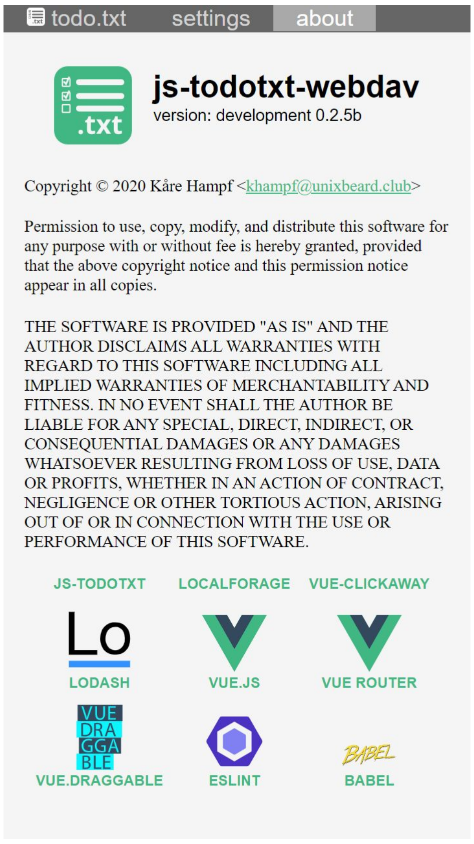
Figure 15. Example of import and use of the js-todotxt library in JavaScript.

Table 1. Description of some of the methods included in the js-todotxt library.

Method	***Description***
getJSONFromTodoTxt(text)	Parses todo.txt into task objects in JSON.
getTodoTxtFromTodoJSON(json)	Parses task objects as JSON to todo.txt-format.
getLineFromTodoTask(task)	Parses a single task object into a line of text in todo.txt format.
taskDifference(tasks1, tasks2)	Produces an array of tasks in the first array missing in the second.
isDone(task)	Returns *true* or *false* depending on whether a task is marked as done.

4.1.2. The js-todotxt-webdav PWA

The user can load the application by visiting the URL address of the hosted task manager application using a web browser. The software license is included in a separate screen as shown in Figure 16. Web browsers supporting the PWA standard such as the Android version of Google Chrome will indicate that the page can be installed locally on the device.



js-todotxt-webdav

version: development 0.2.5b

Copyright © 2020 Kåre Hampf <khampf@unixbeard.club>

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

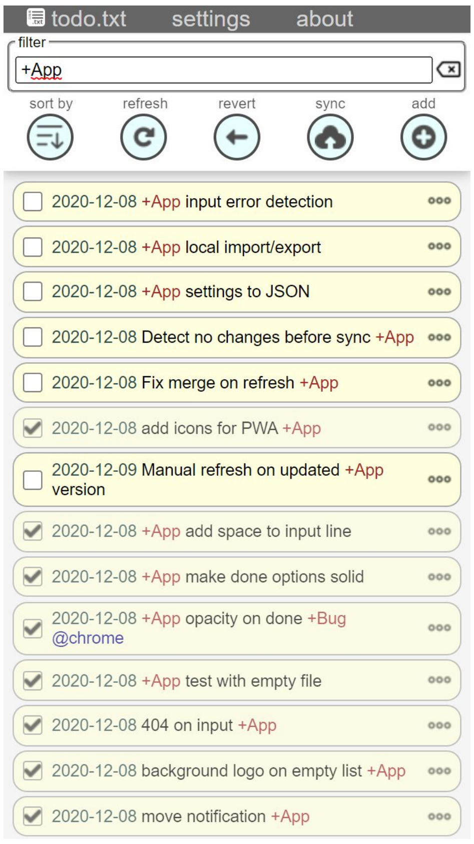
THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Figure 16. The todo.txt application displaying the about screen with the ISC software license.

4.1.2.1. Using the application

Users can enter tasks using the *add* button while viewing the task list on the application home screen and then enter text in the task editor component. Entered tasks are added to the list where they then can be manipulated (edited, marked done, sorted etc).

Interacting with the text input labeled *filter* suggests previously used project- and context-tags as well as manual input of words (or parts of words) for filtering the displayed tasks. Filtering will hide all tasks not containing the specified words as shown in Figure 17.



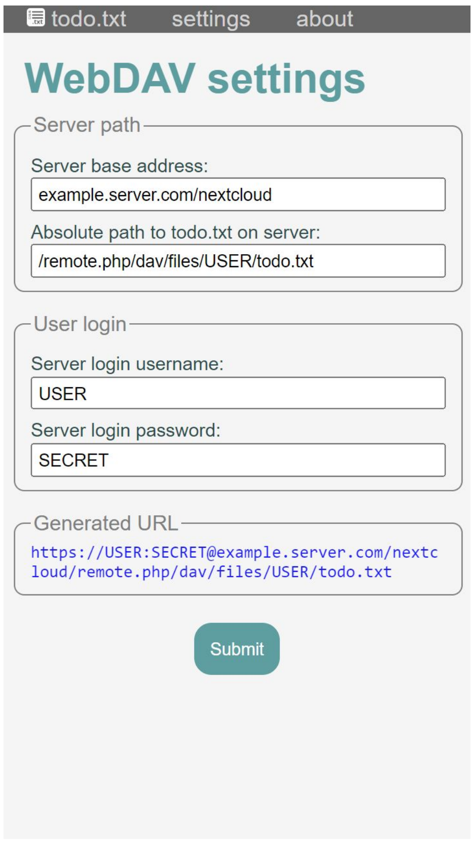
- ☐ 2020-12-08 +App input error detection
- ☐ 2020-12-08 +App local import/export
- ☐ 2020-12-08 +App settings to JSON
- ☐ 2020-12-08 Detect no changes before sync +App
- ☐ 2020-12-08 Fix merge on refresh +App
- ☑ 2020-12-08 add icons for PWA +App
- ☐ 2020-12-09 Manual refresh on updated +App version
- ☑ 2020-12-08 +App add space to input line
- ☑ 2020-12-08 +App make done options solid
- ☑ 2020-12-08 +App opacity on done +Bug @chrome
- ☑ 2020-12-08 +App test with empty file
- ☑ 2020-12-08 404 on input +App
- ☑ 2020-12-08 background logo on empty list +App
- ☑ 2020-12-08 move notification +App

Figure 17. The todo.txt application displaying a list of tasks.

Added tasks remain between application restarts and the application can after the initial load into a browser tab or after local installation be used while being offline and disconnected from the hosting server. The PWA does not require the hosting server to be reachable for normal operation provided the WebDAV server is still reachable and should the WebDAV server not be reachable either, the application will continue to accept tasks for later synchronization. Synchronization between instances of the task manager application running on other devices or with other applications and editors having access to the todo.txt file works as intended.

4.1.2.2. Configuration of WebDAV credentials

The server address and credentials used for connecting with WebDAV for synchronization can be configured in a settings form as shown in Figure 18. The form can be accessed by going to the *settings* tab at the top of the screen.



WebDAV settings

Server path

Server base address: example.server.com/nextcloud

Absolute path to todo.txt on server: /remote.php/dav/files/USER/todo.txt

User login

Server login username: USER

Server login password: SECRET

Generated URL

```
https://USER:SECRET@example.server.com/nextc
loud/remote.php/dav/files/USER/todo.txt
```

Figure 18. The application displaying the settings dialog.

4.1.3. The todotxt Nextcloud application

When the task manager application in pure PWA form became compatible with WebDAV using Nextcloud for the purpose of deployment of the application itself was no longer necessary. Development of the Nextcloud application was halted. The PWA can access and synchronize files in Nextcloud instances without the need for a Nextcloud application, provided the web server CORS configuration allows WebDAV access. As the PWA has a separate interface class for reading *todo.txt* using WebDAV, creating a similar component using the same interface but that integrates with the Nextcloud API instead is still possible.

4.2. Reflections

Working with JavaScript and PWAs has been an interesting journey that started with a teacher telling the class that "JavaScript is the future" which stood in contrast with my past experiences. My own experience of JavaScript at the time was that it was barely useful in verifying input in web page forms. I could not resist the idea of revisiting the language. An abundance of books, articles and web posts about JavaScript exist today and it is easy to get started.

As a language JavaScript offers most of the capabilities of other object-oriented programming languages while enforcing very few rules. The syntax is relaxed, you can for example omit semicolons at the end of most lines of code, it lets you assign whatever you like to variables and instead of a logical true and false it pertains to concepts such as "truthy" or "falsy".

A lot of functionality in JavaScript is not yet completely supported by all interpreters. One example, the "standard modules" introduced in ECMAScript 2015 are supported by all major browsers but not yet fully in Node.js. This leads to confusion over how to actually use JavaScript modules in any form. It has been five years since the standard was introduced, come on! Several frameworks seem to provide the exact same functionality making you question whether you should choose one over the other based on how much their logo appeals to you. One article can describe the best practices to follow when designing a component while the next article describes the exact same method as an anti-pattern. Concepts in JavaScript can be difficult to understand as there is no clearly defined standard practice of how to use them. The upside of this is that when you at last get a grasp of things, nothing is as complicated as it appeared to be at first.

The task manager application is completely usable in its current state and it does what it is supposed to do. Synchronization with a *todo.txt* file stored on a remote server works great. Looking forward, provided development is continued, the application can offer more functionality and provide a more user-friendly interface.

REFERENCES

ECMA. (2017). *The JSON Data Interchange Syntax* (No. ECMA-404). https://www.ecma-international.org/publications/standards/Ecma-404.htm

Gerchev, I. (2018, December 19). *Design Patterns for Communication Between Vue.js Components*. Envato Tuts+; Envato Tuts. https://code.tutsplus.com/tutorials/design-patterns-for-communication-between-vuejs-component--cms-32354

IETF Group, IETF Network, & Others. (2007). *HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)* (RFC 4918). https://www.rfc-editor.org/info/rfc4918

Johnson, J. (2018). *hello-pwa* (Version 1.0.0) [Computer software]. GitHub. https://github.com/jamesjohnson280/hello-pwa

Lodash team and contributors. (2009). *Lodash*. https://lodash.com/

Mozilla and individual contributors. (2013, April). *localForage*. LOCALFORAGE - Offline Storage, Improved. https://localforage.github.io/localForage/

Mozilla and individual contributors. (2019). *Cross-Origin Resource Sharing (CORS)*. MDN web docs. https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Mozilla and individual contributors. (2020a). *Classes*. MDN web docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

Mozilla and individual contributors. (2020b). *File and Directory Entries API*. MDN web docs. https://developer.mozilla.org/en-US/docs/Web/API/File_and_Directory_Entries_API

Mozilla and individual contributors. (2020c). *Progressive web apps (PWAs)* (draft). MDN web docs. https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

Nextcloud GmbH. (2020). *Tutorial — Nextcloud latest Developer Manual latest documentation* (Version 19.0). https://docs.nextcloud.com/server/latest/developer_manual/app_development/tutorial.html

Paltoglou, A., Zafeiris, V. E., Giakoumakis, E. A., & Diamantidis, N. A. (2018). Automated refactoring of client-side JavaScript code to ES6 modules. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 402–412.

Rischpater, R. (2015). *JavaScript JSON Cookbook*. Packt Publishing Ltd.

Rojas, C. (2019). *Building Progressive Web Applications with Vue.js: Reliable, Fast, and Engaging Apps with Vue.js*. Apress.

Rollup contributors. (2020). *Rollup* (Version 2.33.3). https://rollupjs.org/guide/en/

Russell, A. (2015, June 15). *Progressive Web Apps: Escaping Tabs Without Losing Our Soul*. Infrequently Noted. https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/

Salnikov, M. (2017, July 3). *progressive-web-apps-logo*. https://github.com/webmaxru/progressive-web-apps-logo/issues/3

Schneider, G. M., & Gersting, J. L. (2018). *Invitation to Computer Science*. Cengage Learning.

Sheppard, D. (2017). *Beginning Progressive Web App Development: Creating a Native App Experience on the Web*. Apress.

Stefanov, S., & Sharma, K. C. (2013). *Object-Oriented JavaScript*. Packt Pub Limited.

Trapani, G. (n.d.). *Todo.txt*. Retrieved November 22, 2020, from http://todotxt.org/

Trapani, G. (2006, May 12). *Geek to Live: Reader-written todo.txt manager*. Lifehacker. https://lifehacker.com/geek-to-live-reader-written-todo-txt-manager-173018

Web Design and Applications - W3C. (n.d.). Retrieved November 20, 2020, from https://www.w3.org/standards/webdesign/

Webpack contributors. (n.d.). *Why webpack* (Version 5). Retrieved November 22, 2020, from https://webpack.js.org/concepts/why-webpack/

Wikipedia contributors. (2020a, October 28). *Ecma International*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Ecma_International

Wikipedia contributors. (2020b, November 4). *Object-oriented programming*. Wikipedia, The Free

Encyclopedia. https://en.wikipedia.org/wiki/Object-oriented_programming

Wikipedia contributors. (2020c, November 19). *Progressive web application*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Progressive_web_application

Wikipedia contributors. (2020d, December 7). *Web browser*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Web_browser

Wikipedia contributors. (2020e, December 8). *Cross-platform software*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Cross-platform_software

Williams, G. (2017). *JavaScript for Microcontrollers*. Espruino. https://www.espruino.com/

APPENDIX

Definitions

API	Application Programming Interface
CORS	Cross-Origin Resource Sharing, a security enforcement technique[29]
CSS	Cascading Style Sheets - styling of HTML (W3C)
DOM	Document Object Model (WHATWG, W3C)
ECMA	European Computer Manufacturers Association[30]
ES6	ECMA-262[31], the ECMAScript standard 6th revision, ECMAScript 2015
ESM	ECMAScript Modules standardised in ECMA-262
FTP	File Transfer Protocol (RFC 959[32])
HTML	Hypertext Markup Language - the markup defining elements in web pages (WHATWG[33])
HTTP	Hypertext Transfer Protocol (RFC 7540[34])
IDE	Integrated Development Environment
ISO/IEC	International Standards Organization/Electrotechnical Commission[35]
Java	A class-based object-oriented programming language (Oracle)
JavaScript	Scripting language conforming to the ECMAScript standard
JSON	JavaScript Object Notation file/interchange format (ECMA-404[36])
MVP	Model-view-presenter software design pattern
Nextcloud	Server software to provide users access to files and applications[37]

[29] https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
[30] https://www.ecma-international.org/
[31] https://www.ecma-international.org/ecma-262/6.0/
[32] https://www.rfc-editor.org/info/rfc959
[33] https://html.spec.whatwg.org/
[34] https://www.rfc-editor.org/info/rfc7540
[35] https://www.iso.org/home.html
[36] https://www.ecma-international.org/publications/standards/Ecma-404.htm
[37] https://nextcloud.com

OOP	Object Oriented Programming
PHP	Scripting language evaluated and run inside web servers[38] (Zend Tech.)
PWA	Progressive Web Application with native app-like UI
REST	Representational State Transfer (REST)
SVG	Scalable Vector Graphics (W3C)
SQL	Structured Query Language used with relational databases (ISO/IEC[39])
todo.txt	Structured plaintext format for listing tasks together with metadata
UI	User Interface
W3C	World Wide Web Consortium[40]
WebDAV	World Wide Web Distributed Authoring and Versioning (RFC 4918[41])

[38] https://www.php.net/
[39] https://www.iso.org/standard/63555.html
[40] https://www.w3.org/
[41] https://www.rfc-editor.org/info/rfc4918