Local Web Application Development
Framework Proposal and Demonstration on Two Case Studies
Extended Abstract

Diogo Miguel de Mello Caldeira Reis Almiro

Instituto Superior Técnico, Universidade de Lisboa, Portugal
diogo.almiro@tecnico.ulisboa.pt

Abstract Web technology has reached a penetration that makes it relevant not only for web applications but also for what used to be the classic domain of desktop applications. The challenge of this project is to propose a reference architecture that combines attributes of desktop applications and attributes of web technology, recommending a solution using mature technology. There are two real-life case studies to demonstrate and validate this technology. Both of them require the management of documents and the execution of optical character recognition. The first case study relates to the workflow when archiving historical documents on the Archives and Records Management Section of the United Nations. The second case study relates to the workflow of making decisions by judges of the "Supremo Tribunal de Justiça". This project starts by introducing some core concepts and requirements for the case studies. It then analyses the state of the art of technologies related to the server-side of an application. Instead of introducing completely new technology, this project introduces an expansion from a particular technology from the analysis. The chosen technology is Express, and the project describes this expansion. It also demonstrates the usage of the proposed solution by implementing small applications for the case studies.

Keywords: Web Technology · Hybrid Application · Framework · Express · NodeJS · OCR

1 Introduction

The so-called "web technology" has reached a penetration that makes it relevant not only for real web applications but also for what used to be the classic domain of "desktop applications", meaning applications intended to execute in a single personal computer by an individual user. Therefore, the challenge of this project was to propose a reference architecture for this kind of scenario, recommending mature technology and demonstrating it.

This work relates to the workflows from two other projects, ARMS and IRIS. These are used as case studies. These relate because both are exploring new technologies to digitise and process optical character recognition (OCR) in an administrative environment of documents.

Although the primary objective is to know the value of the technology proposed, it should also implement the case studies applications with the solution to be the most valuable to the end-user.

2 Core Concepts and State of the art Technology

A software application, or application for short, is a program created to perform specific tasks. It can be classified regarding its installation method into three categories: native application, web application and hybrid application. Regarding the application's permissions, they are directly dependent on the application's classification. Native and hybrid are very trusted applications by the OS with many permissions, while web applications are run inside the browser's sandbox [3].

Regarding the development of a hybrid application, it can be seen as developing a web application using the client-server paradigm. The positive aspects are that it is simpler to develop the user's interface, use the browser as support, and have full access to the system like a native application on the back-end [5].

A framework is an artefact that helps solve a group of problems. It improves development efficiency and reduces cost by implementing several functions or classes to provide the typical behaviour for similar problems [8]. Flask[1] and Express[2] are examples of back-end framework, as they implement server functionalities, while Bootstrap[3] is an example of a front-end framework, as it provides CSS styles for well-defined HTML classes.

A test-driven development approach (TDD) can be used to validate the development process, ensuring the quality of the solution proposed and that all the behaviours were the expected ones [2, 6]. It is common to perform small local tests, unit tests on a TDD approach. There are frameworks to facilitate their creation, such as Mocha[4]. Another perspective is the end to end testing (E2E), where an application is fully tested. A framework to perform E2E is Cypress[5].

Regarding data management, databases are a collection of data structured according to a schema. There are several data storage paradigms[6]. Regardless of its structure, the database usually exists in a separate server accessed by the web application.

3 Requirements

This project aimed to create a framework that helps generate hybrid applications, specifically local-only web applications [7]. Using the client-server paradigm,

[1] See palletsprojects.com/p/flask/
[2] See expressjs.com/
[3] See getbootstrap.com/
[4] See mochajs.org/
[5] See www.cypress.io/
[6] See fireship.io/lessons/top-seven-database-paradigms/

where the browser is the user interface, and the server has access to local data and programs having also permissions to download extra dependencies. Note that a hybrid application could be extended to a web application for other users inside the private network.

It created some user scenarios for the ARMS end-user, the IRIS end used, the administrator and the developer to understand better what functionalities the framework should have and each case study's application should have:

- For ARMS application:
1. Select a folder with TIFF files to process.
2. Execute a bulk OCR to the files.
3. Notify the user.
4. See OCR metadata.
5. Edit OCR results.

- For IRIS application:
1. Select a PDF document to use.
2. See the document and annotations.
3. Annotate the document.
4. Correct the OCR.
5. Synchronise an external editor to the application.
6. Export an archived document.
7. Import an archived document.

4 Analysis of the State of the Art Technology

The project started by defining some criteria to evalutate frameworks: "Popularity & Community", "Maintainability & Support", "Documentation & Examples", "Use Cases", "License", "Learning Curve".

The first criteria defined the search space: the most popular back-end frameworks of 2020, based on StackOverflow's survey [7]. Which yielded: Express, ASP.NET Core, Spring, Flask, Laravel and Ruby on Rails.

Their release date varies a lot. However, all are actively updated on Github. They are also very popular, and as such, they are well documented or at least the strong community can help developers [8]. A lot of big companies use them on a day to day basis. Every framework is open source, and, between the licenses, there are no significant differences that affect the project. All grant modification, distribution, commercial and private use, no warranty and no liability [8].

The main differentiation point is the "Learning Curve", which connects directly to the programming language used. Another factor that affects it negatively is if the framework uses convention over configuration. For instance, by forcing a project structure, the framework can be harder to learn [9]. Only Express and Flask are not "opinioated".

Regarding the database, using an external database would mean having an internet connection to the database server or having a different server for the database consistently running in the background of the user's device, spending

[7] See insights.stackoverflow.com/survey/2020
[8] See choosealicense.com/licenses/bsd-2-clause/
[9] See facilethings.com/blog/en/convention-over-configuration and devopedia.org/convention-over-configuration

more resources than needed. Using a particular type of database, an embedded one can exclude the previous problem. With them, there is no external server. Instead, the database coexists inside the webserver application. It can also remove the extra processing time of continuously running a database and have a data connection between servers using an embedded database [1,4].

Electron[10] is a framework that allows developers to build native applications using web technologies. Among other problems like user experience, when a user installs an Electron application, it also installs a built-in version of the Chromium browser, making the application require more storage to install. Using Electron would also prevent having a distributed application within a network as it does not automatically run a server.

5 Solution Proposed

All frameworks are relatively alike, with similar popularity and community. However, Express's "unopinionated" characteristic allows it to have a more flexible project structure. Since there is a need to configure the application for each use case, it sounds more reasonable to be able to configure the project as well. The project also does not require any specific language for the server-side. Albeit, Flask would also give the configuration freedom. The fact that Express extends the functionality of NodeJS allows to have a consistent language (JavaScript) in both the front-end and back-end of the project and a consequent JSON communication between both layers. Another factor to use the Express framework is the community around the NodeJS runtime and JavaScript language.

After reviewing the architecture, see figure 1, the approach on the server-side was changed to a more modular approach to the problem, segregating the application's components and having an unopinionated view over the database usage, enabling the reuse of components for other applications.

6 Implementation

The framework is structured into an "app" and a "component" package. The first package defines the `WebfocusApp` class, while the second defines the class `WebfocusComponent`. There are also three packages with reusable components: "send-mail", "util", and "tray". Use `$ npm install @webfocus/<package-name>` to install and use them. See figure 2 for an overview of the framework packages and classes. With this structure, it is possible to implement components without requiring the whole application package.

Next follows a short description of the three main reusable components implemented.

Util: Some client-side and server-side generic utilities. It sets a property `hidden` to true soo that the end-user does not see it. It provides the static files

[10] See www.electronjs.org/

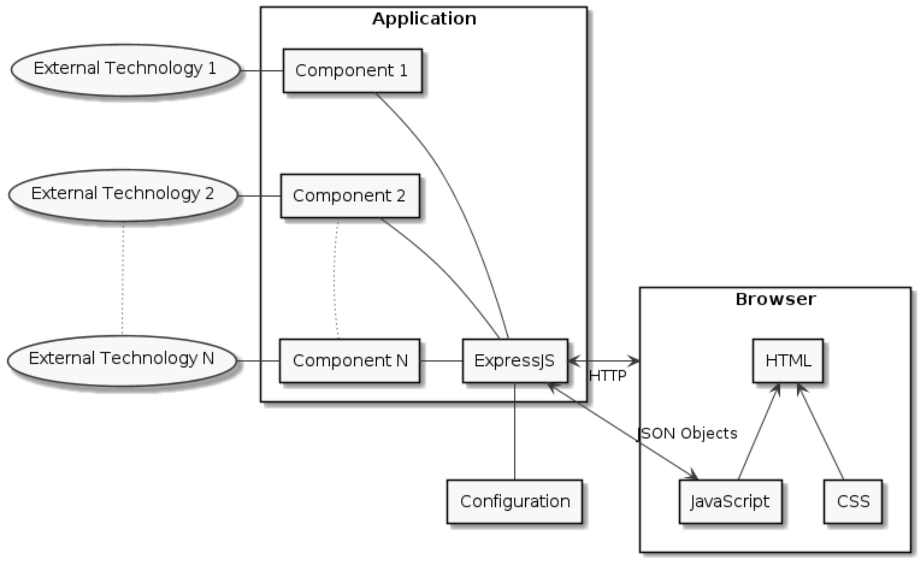

Figure 1: Reviewed framework architecture.

`submit-json.js`, `pagination.js` and `inline-fecth.js` to the browser, all of them depending on the previously mentioned `/wefocus-static/js/fetch.js` file. It also provides on the server-side the functions `serversideevents` and `pagination`.

Send Mail: It allows configuring an email server in order to be able to send emails to the end-user. It uses the `nodemailer` package and defines the function on the app instance `sendMail` so that other components can also send emails if defined. Its `index.pug` is a submittable form to update the server configuration locally.

Tray: It allows the application to have an icon and menu on the system tray of the end-user. It uses the `ctray` package that this project implemented outside the framework's scope as a nice-to-have feature. Its `index.pug` is a repetition of the actions on the system tray in case it is not available.

A TDD approach was used to ensure the behaviour of the "app" and "component" packages as they are the core of the framework. E2E tests were also done on a sample application, ensuring the behaviours of the application as a whole.

7 Demonstration

The framework was applied to several real-life implementations related to the case studies.

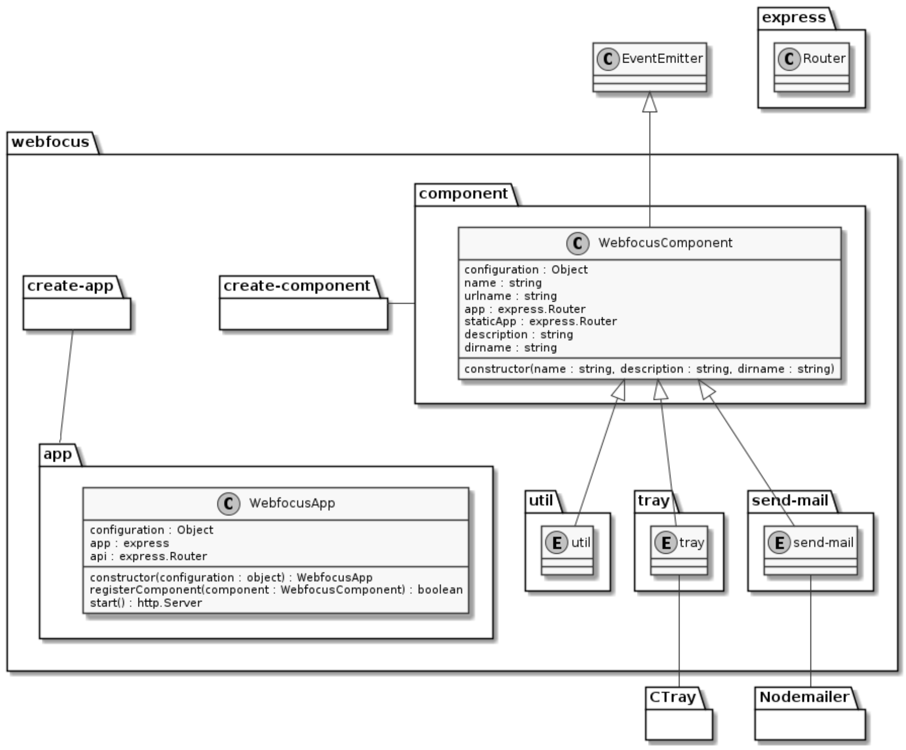

Figure 2: Webfocus Framework overview.

The first application from the early stages of the framework featured the connection to an embedded database.

Regarding IRIS, an application was created to explore the DGSI[11] public database. Beforehand the DGSI database was collected to local files. This application allows us to compare the summary and the full text of each process. It also allows seeing other metadata related to the process. This implementation inspired a more profound report over the DGSI database, as abnormal fields were found.

Another IRIS application showcases the integration between the text editor of DOCX files and the application. It demonstrates that it is possible to detect and update in real-time changes in the external technology.

The first ARMS implementation was the proof of concept that shows it could run the external technology and create a ZIP file with the OCR result.

The second ARMS implementation does not implement everything from the proof of concept, but it demonstrates that it can control the Docker Engine.

Table 1 specifies all the use cases implemented from the previous descriptions.

[11] See http://www.dgsi.pt/jstj.nsf/

Table 1: List of the use cases.
■- Implemented, ☒- Partially Implemented

☐	Case Study	Use Case
	ARMS	
■		Select a TIFF files' location.
■		Define OCR settings.
■		Execute a bulk OCR on a location.
■		Notify the user.
☒		See TIFF files.
☐		See OCR metadata.
☒		Edit OCR results.
☐		Regenerate a PDF from manually edited OCR.
	IRIS	
☐		Select a PDF document to use.
☐		See the document and annotations.
☐		Exectue OCR on demand.
☐		Annotate the document.
☐		Correct the OCR.
☒		Synchronise an external editor to the application.
☐		Export an archived document.
☐		Import an archived document.

8 Conclusions

The challenge of this project was to propose a reference architecture for the use of web technologies in the desktop domain. Precisely, it ought to improve workflows in an administrative environment of two case studies: digitising historical documents on ARMS and analysing legal documents on a top national court.

After introducing some basic concepts, aimed solution and the application characteristics were described according to the case studies. Afterwards, technologies that could help solve the problem were analysed, reaching a reasoned ground to formulate a solution and a proposed architecture in more detail. Architecture that was implemented afterwards alongside several use cases to demonstrate it.

References

1. A. Adamanskiy and A. Denisov. Ejdb - embedded json database engine. In *2013 Fourth World Congress on Software Engineering*, pages 161–164, 2013.
2. Cristian Augusto. Efficient test execution in end to end testing : Resource optimization in end to end testing through a smart resource characterization and orchestration. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 152–154, 2020.

3. Adrienne Porter Felt, Kate Greenwood, and David Wagner. The effectiveness of application permissions. WebApps'11, page 7, USA, 2011. USENIX Association.
4. L. Junyan, X. Shiguo, and L. Yijie. Application research of embedded database sqlite. In *2009 International Forum on Information Technology and Applications*, volume 2, pages 539–543, 2009.
5. G. Kappel, B. Pröll, S. Reich, and W. Retschitzegger. *Web Engineering: The Discipline of Systematic Development of Web Applications.* Wiley, 2006.
6. R. Paul. End-to-end integration testing. In *Proceedings Second Asia-Pacific Conference on Quality Software*, pages 211–220, 2001.
7. Peter Seebach. Develop web applications for local use. *IBM, developerWorks*, 2007, accessed October 2020.
8. Leonor Teixeira, Ana Raquel Xambre, Helena Alvelos, Nelson Filipe, and Ana Luísa Ramos. Selecting an open-source framework: A practical case based on software development for sensory analysis. *Procedia Computer Science*, 64:1057 – 1064, 2015.