Volume 9, Special Issue No. 3, May 2018

ISBN: 978-93-5311-910-2

International Journal of Advanced Research in Computer Science
(ISSN: 0976-5697)
CONFERENCE PAPER

Available Online at www.ijarcs.info

OpenRAP: A Distributed, Scalable, Offline CDN

Sriram V Ramaswamy
School of Computing and Information Technology
REVA University
Bengaluru, India
sriram.rmswmy@gmail.com

Sumukha K V
School of Computing and Information Technology
REVA University
Bengaluru, India
kvsumukha@gmail.com

Ravish Ahmad
School of Computing and Information Technology
REVA University
Bengaluru, India
ravishahmad16@gmail.com

Shah Abdul Ghani
School of Computing and Information Technology
REVA University
Bengaluru, India
gshahabdul@gmail.com

Kiran M
School of Computing and Information Technology
REVA University
Bengaluru, India
kiranm@reva.edu.in

Abstract—This paper aims to implement a fully functional localized scalable Content Delivery Network (CDN) that allows for seamless content distribution without any need for internet access from a cheap, low-powered, headless IoT device. The software can be run out of the box and can be used with minimal technical knowledge on the side of the user. This state-of-the-art proposed system can be implemented in many fields that require streaming and/or media download services, especially those that are highly localized in nature and do not need continuous connectivity with the internet.

Keywords—Content Delivery Network (CDN), NodeJS Server, React App, SQLite3, Internet of Things (IoT)

I. LITERATURE REVIEW

IoT networks today are very prevalent, bringing in ubiquitous computing to the masses. With a projected amount of connected devices to reach 24 billion by 2020 [1], it becomes important for decentralized networks to have proper footing within themselves so as to reduce workload on a few centrally located servers.

On the other side of the spectrum, the reach of IoT networks hasn't been very promising. People do not have access to a decent cellular network a few kilometers outside an urban agglomeration. Studies show that only 9% of India has access to mobile internet, and cloud-based IoT networks, coupled with expensive hardware, can make proper connectivity just a dream for many [3]. Therefore, it becomes important that any solution that decides to solve this problem needs to be cost-effective and fully functional.

The rise of FOSS and Linux-based operating systems has enabled nearly seamless transitioning between IoT devices capable of running their own operating systems. What difference remains, is mainly about which mini-computer to use. The following table provides an easy comparison of commonly available boards, and their features.

TABLE I. DIFFERENT IOT BOARDS

Boards	Features
Raspberry Pi	1GB RAM, 1.2GHz CPU
Beaglebone Black	512MB RAM, 3D GPU
ASUS Tinkerboard	2 GB RAM, 1.8 GHz CPU
Qualcomm Dragonboard	1GB RAM, 1.2GHz CPU

These boards are available with most electronics deals and can easily be purchased online. The fact that these boards do not cost more than $100 implies a very good economic viability of these devices when compared to a full-sized server. Furthermore, all these boards can run a version of Debian/Ubuntu compiled specifically for them, wherein the operating system has an optional desktop interface and can remotely be connected via SSH.

The common server frameworks are as follows [6][7][8][9][10][11]:

- NGINX
- Apache

- Django
- NodeJS
- Flask
- Drupal

Of all these, NodeJS is an excellent choice due to its relative success and popularity among developers. While other frameworks like Drupal [a self-styled CDN network] and Django [a full-stack web framework] are popular, NodeJS is relatively easy to use and easy to deploy.

II. ARCHITECTURE OVERVIEW

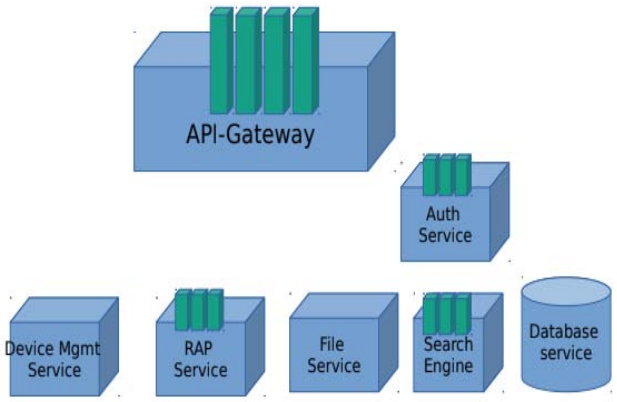

Fig. 1. The Basic Architecture of Our State-of-the-Art Proposed System

The architecture, as described above, forms the core functionality of OpenRAP. Plugins [described by the green vertical cuboids], can be implemented on top of these services as and when needed. Their functionalities are as follows:

A. API Gateway

The API Gateway contains all the routes that shall be implemented by the device in order to function appropriately. Those building plugins will need to define routes in the API Gateway so as to ensure accessibility of the newer functionalities. The APIs can be defined in a simple `routes.js` file.

B. RAP Service

The RAP Service contains the core functionality for the device, including file management, user management, etc.

C. File Service

The File Service provides interfacing for file operations and can be extended by means of plugins to, for example, serve files by means of a custom behavior.

D. Search Engine

The Search Engine is based on BleveSearch and can perform fuzzy searches in order to serve files and get data more easily.

E. Auth Service

The Auth Service provides authentication and authorization functions for the user. While it currently performs very less operations, it can be extended by means of plugins to perform authentication, authorization, token verification, etc. or even extend common tools like OAuth and Google Sign-In.

F. Database Service

The Database Service provides wrapper functions for common DBMS operations.

III. TECHNOLOGY STACK

The proposed system is a full-stack JavaScript project that can run on any platform and architecture. This state-of-the-art system can be divided into the following sections:

A. Backend

The backend is a NodeJS server that exposes a number of API routes to the user. These routes can be interfaced by the default front-end or any other application capable of making such HTTP requests and parsing the response data. This is a full-fledged API server; all routes return JSON objects which denote whether the proposed system was successful, and any corresponding data and messages.

The backend also makes use of a database to record user activity. It uses a MYSQL database along with a wrapper that allows for seamless integration with the NodeJS backend. The wrapper is available in the form of an SDK which can be used by a developer to create plugins.

B. Frontend

The default frontend is available in the form a basic React App that allows for essential device management services such as file and user management. The frontend is completely independent of the backend and thus can be replaced with a custom frontend for any extensibility by the user. The advantages of using React include extensive documentation, a large amount of available third-party modules and JSX functionality.

C. SDKs

The proposed system comes with prebuilt SDKs that allow for easy, generic interfacing with the backend. The SDKs contain functions that in turn process the request and prepare an appropriate response for the user to use. These SDKs can be interfaced by the plugin writers as well in order to easily extend features of the device. The SDKs available are:

a. DB SDK

This is the database SDK that allows for the plugin writer to implement DBMS operations. This works best with a MySQL backend but can also be used for a lightweight DBMS such as SQLite3. The plugin writer need not write logic by themselves for a simple task; the SDK provides a simple promise-based interface for the plugin writer to use.

b. File SDK

The File SDK allows for easy file management operations, which run complex code implementing middleware and NodeJS's 'fs' module. As the underlying code is a mixture of promise-based and callback-based code, it can cause a lot of confusion and unnecessary effort for someone who needs to write a simple plugin and may not be very experienced with the JavaScript Environment. The File SDK provides basic functionalities that include, but are not limited to, copying and moving files, creating and deleting folders and uploading files. The File SDK forms the core part of the file management module of the device.

c. Search SDK

The Search SDK is a novelty addition to the module that is exclusively built for plugin use. The Search SDK acts as a wrapper around BleveSearch, which is an open-source goLang library built on the ElasticSearch platform. The Search SDK performs internal API calls to the BleveSearch module, thereby acting as a middleware to the search functionality. The core module has no use case for the Search SDK, but this can be extended by means of plugins.

d. Plugins

Plugins are user-written code that allow for extensibility of the CDN's features. These plugins can be written in any language but work best when homogenous with the core modules, eliminating any unwanted need for trans-piling or interfacing. Plugins can use the SDKs available or use their own logic to get things working.

IV. WORKING

The state-of-the-art proposed system acts as a typical server with middleware and plugin support. Due to the heavily modular nature of the CDN, it can also be used in a distributed system with minimal alterations to the server. Those intending to scale the CDN up and host it in an AWS instance can also do it without much hassle. The proposed system is platform-independent and therefore can be easily hosted anywhere.

The basic workflow of the code is as follows:

A. The user performs an action.
B. The React component registers that action and calls the appropriate function in `<component>.actions.js` while optionally using any data it might require from the state or variables present in `<component>.reducer.js`.
C. The function present in <component>.actions.js calls the appropriate HTTP route using Axios.
D. The request is caught by the backend server, where it verifies it against all `routes.js` files. Once a match is found, it calls the corresponding function in the appropriate `controller.js` file.
E. The method present in the `controller.js` file calls the appropriate SDK/Helper functions and prepares the response object.

V. PROMISES

Promise-based functions have a unique structure when compared to other callback-based functions in that they allow for segregated logic in case of successful and unsuccessful executions by means of `resolve` and `reject` attributes. A method that typically executes perfectly "resolves" a value, while a method that fails to execute perfectly "rejects" the reason. This is handled separately by the function that calls it, at its `then`attribute[5].

Therefore, a promise-based function can be expressed as follows:

```
letfunc = (params) => {
    let defer = q.defer(); //q is an object of module Q
    doSomething(someOtherParams, (err, data) => {
    if (err) {
    returndefer.reject(err);
    } else {
    returndefer.resolve(data);
    }
    }
    returndefer.promise;
}
```

The structure of the calling function is as follows:

```
letfunc = (params) => {
let defer = q.defer();
    callAPromisedFunction(someOtherParams)
    .then(value => {
     //Process data
    returnanotherPromisedFunction(stillMoreParams);
    }).then(value => {
    returnyetAnotherPromisedFunction(yetMoreParams);
    }).then(value => {
    .
    .
    .
    returndefer.resolve(someValue);
    }).catch(e => {
    console.log(e);
    returndefer.reject(e);
    });
}
```

Promises, especially in NodeJS, are designed to combat what is colloquially called "callback hell", i.e. asynchronous code that is a part of a synchronous execution chain. As the primary method of asynchronous code handling is to use callbacks, anyone using asynchronous code repeatedly can find the code very hard to maintain.

By employing promises, the code's readability can considerably increase, by a means called "flattening the pyramid" or "promise chaining". This allows code to maintain the same hierarchical level across execution[5].

V. DETAILED SDK FUNCTIONALITY

This section outlines the detailed functionality the SDKs offer.

A. File SDK

The functionalities of File SDK are as follows:

a. readFile: Reads a file
b. writeFile: Writes data to a file
c. deleteFile: Deletes a file
d. deleteDir: Deletes a dir
e. copy: Recursively copies a file/folder
f. move: Recursively moves a file/folder
g. readdir: Reads a directory
h. extractZip: Extracts a ZIP
i. extractTar: Extracts a TAR
j. createTar: creates a tar.gz file

B. DB SDK
   a. createDatabase: Creates a database
   b. deleteDatabase: Deletes a database
   c. createTable: Creates a table in a database
   d. listTables: Lists all tables in the database
   e. deleteTable: Deletes a table in a database
   f. addRecord: Adds a new record to the table
   g. readRecords: Reads available records in a table

C. Search SDK
   a. init: Initializes the search and DB directories, while loading any existing data
   b. createIndex: Creates an index
   c. deleteIndex: Deletes an index
   d. getAllIndices: Gets a list of available indices
   e. addDocument: Adds a document to an existing index
   f. deleteDocument: Deletes the document in the index
   g. getDocument: Retrieves a document from the index
   h. count: Gets the count of documents in an index
   i. search: Searches for a query in the index

VI. RAP SERVICES

The RAP services offered by the device are as follows:

- The RAP service allows a superuser to modify the permissions of, delete, or create a new user. These permissions limit the ability of the user to perform administrative tasks to a set of well-defined roles.
- The dashboard contains data pertaining to a device's health and statistics such as the number of users connected, the version number, etc. This is by default available for all users. The statistics include disk, RAM and processor usage.
- The upgrade screen allows the user to upgrade the device by uploading a `.tgz` file.
- The File Management window employs the File SDK and allows the user to upload, modify and delete files in the device through the browser. It also provides an interface for folder creation and updation.
- The SSID Management Component allows the user to manage the SSID the device broadcasts.
- The Captive Portal Management Component modifies the default captive portal that users see on connecting to the device. This allows for custom messages and download links to be displayed instead of the default Apache2 page.

VII. USE CASES

The proposed system has a diverse set of use cases.

- The proposed system can be used to bring education to rural schools in remote and tribal areas with very less internet connectivity, allowing students to have a more immersive learning experience. Lessons can be preloaded into the device which can then be used by the students in their respective smartphones/tablets.
- Digital classrooms in urban areas can be decentralized by means of using OpenRAP for every classroom. This completely avoids the problem of a centralized server crashing or breaking down due to heavy load.
- Examinations can easily be conducted as a unique login can be provided to the students by building a plugin over the Auth service which can provide more unique examinations and a higher degree of transparency.
- Buses can now contain a device that streams movies and other data on-demand to customers who require it, thereby allowing other passengers to enjoy their ride peacefully.
- Airplanes can have different devices for different sections of the aircraft, thereby allowing people to register their seats and book meals as and when they require them.
- Operas and stage plays can allow the audience to download and view the script/schedule of the play(s), leading to an even more immersive experience for theatre-goers.
- Banks can now have an automatic, phone-based token system instead of requiring customers to use the queue.
- Locally available services can now be automatically obtained and citizens can be directed to use them.

VIII. FUTURE ENHANCEMENTS

Possible improvements of this paper can include:

- Allowing for massively scaled application using software such as Docker and Kubernetes. As the proposed system is extremely modular, newer modules and plugins can be added and removed as and when needed.
- Integration with technologies such as Blockchain in order to locally store data securely, further increasing utility in sectors such as banking and data storage.
- By deploying the software into cloud-based instances such as Amazon EC2, one can allow for millions of similar clusters in areas with high connectivity.

REFERENCES

[1] Arxiv.org, 2018. [Online]. Available: https://arxiv.org/pdf/1207.0203.pdf?ref=theiotlist. [Accessed: 20-Apr- 2018].
[2] [Online]. Available: https://www.electromaker.io/blog/article/10-best-raspberry-pi-alternatives. [Accessed: 20- Apr- 2018].
[3] [Online]. Available: https://www.gagetsnow.com/tech-news/Only-9-of-rural-India-has- access-to-mobile-internet-Report/articleshow/50840296.cms. [Accessed: 20- Apr- 2018].
[4] Raspberry Pi 3 Model B - Raspberry Pi", Raspberry Pi, 2018. [Online].Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed: 20- Apr- 2018].
[5] Promise", MDN Web Docs, 2018. [On-line]. Available:https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/GlobalObjects/Promise.[Accessed:20 − Apr − 2018].
[6] The Web framework for perfectionists with deadlines — Django", Djan-goproject.com, 2018. [Online]. Available: https://www.djangoproject.com/.[Accessed: 20- Apr- 2018].
[7] NGINX — High Performance Load Balancer, Web Server, Reverse Proxy",NGINX, 2018. [Online]. Available: https://www.nginx.com/. [Accessed:20- Apr- 2018].
[8] Node.js", Node.js, 2018. [Online]. Available: https://nodejs.org/en/. [Accessed: 20- Apr- 2018].
[9] Welcome — Flask (A Python Microframework)", Flask.pocoo.org, 2018.[Online]. Available: http://flask.pocoo.org/. [Accessed: 20- Apr- 2018].
[10] Drupal - Open Source CMS", Drupal.org, 2018. [Online]. Available: https://www.drupal.org/. [Accessed: 20- Apr- 2018].
[11] Welcome! - The Apache HTTP Server Project", httpd.apache.org, 2018. [Online]. Available: https://httpd.apache.org/. [Accessed: 20-Apr- 2018].