Literature Review

Unhosted email Client

CS 4202 Research & Development Project

Supervised by: Prof. Gihan Dias

Team members:	Premathunga, D.D.R.C.	100409M
	Sureshkumar K.	100527X
	Erlilverl K.	100123F
	Waidyarathna U.H.S.A.	100563D

Contents

1. Introduction

1.1. Unhosted web apps

Unhosted web architecture is a new emerging web architecture which has gained a lot of attention in many communities around the world.

1.2. Definition of unhosted web applications

“Also known as "serverless", "client-side", or "static" web apps, unhosted web apps do not send your user data to their server. Either you connect your own server at runtime, or your data stays within the browser.”[1]

Idea behind this definition is you receive the app's source code over the internet or somewhere else. Application logic and front end web application totally runs on the user’s browser. The backend of the web application is swappable, which means it is not tied up with any particular backend server.

This lets the users to choose his own backend as his wish, instead of vendors provide the both web application and backend. This breaks web 2.0's monopoly platforms. This flexibility lets the front end developers to focus on UI/UX and lets backend developers to build more flexible back-ends.

An example of this are Dropbox-based apps [2]. Developers can get an API key from Dropbox account, and develop a JavaScript app, which stores all application data on the user’s Dropbox account. Since Dropbox API supports CORS headers, the in-browser app can make a cross-origin AJAX request directly to Dropbox. Apart from this scenario, a RESTful web service based API is not the only way to develop an unhosted app. If the backend server can talk any protocols other than HTTP and browser can also talk that protocol we can leverage the application to speak that protocol as well.

The typical web stack of a web application is data, protocol, applogic, interface. All layers are highly coupled with each other. Unhosted architecture decouples web apps from their backend storage on a per user level. Protocol layer and data layer get separate from applogic and interface

1.3. Unhosted email client

Unhosted email clients comes into play along with PGP and getting rid of email monopoly existed in today. Today almost all the email clients are hosted. If we take Gmail, it is hosted within Google domain. Yahoo mail client is hosted within yahoo mail domain. We can't use Gmail application with yahoo mails.

By applying unhosted web architecture to email clients we can separate the emails (data) and email client (web application). The problem with yahoo mail, outlook and Gmail is each application has its own features. Some features and user interfaces are better than others. But only considering features of the mail client, users cannot choose a mail client because they only allow mail facilities to their own mail servers. This has narrowed down the freedom of the user. With unhosted email clients, users are no more tied up with such constraints. Users can choose unhosted email client which are written in HTML5, CSS and javascript considering any parameters they wish and use it with any backend that means either with their own mail servers or with other vender mail servers. Since these are typical web application this could run on any platforms.

2. Related works (Papers)

2.1. Linked Data Platform for Web Applications by Joe Presbrey[3]

This research paper propose somewhat very similar to the unhosted web architecture. Instead it presents a decentralized web app platform named Linked Data which eliminates vendor lock-in, and separates user data from web apps giving users control over their data and where it's stored, independent of choice of application.

This architecture ends proliferation of proprietary Web APIs, by decoupling web application logic from data across a standardized HTTP API. Users of these applications gets more highly enriched and interlinked databases than that of any Web API, while retaining control of their own data, and freedom from vendor lock-in.

The platform is a client-server system, in which the servers are based on HTTP

Servers, but enhanced Querying, Updates, and Security.

The Linked Data Platform is a RESTful HTTP service like many other Web APIs

Today. But instead of structured, all data exchanged between peers is modeled using W3C Linked Data standards.

This has provided 3 server implementations using PHP, python and go programming languages. But in unhosted web applications we don't need to install any additional servers. All the communication happens through the standard communication protocols. So here we don't have a need of additional data structures like Linked data structure.

2.2. Javascript IMAP email client for browsers [4]

This project has released a proof of concept for an IMAP client library written in java script which runs as a Chrome app. This has written on top of node.js using node's IMAP client library. To open a raw TCP connection to a server it has used Chrome's socket API. To test this you can either download it from the Chrome app store or install it manually.

Chrome apps can only run on Google Chrome platform. Generally a browser doesn't allow a web page to open up a TCP socket connection to a server due to security concerns. But Chrome apps can open a TCP connection to a server by using socket API provided by Chrome browser. Most of the popular browsers provide browser APIs to open a TCP connection to a server though it is not a W3C standard.

This project has mentioned several future works to be done. They expect to extend this so that it could work on any browser. So in our solution we have left the creating TCP connection part to the browser extension and other application logic and use interfaces has moved in to the web application. So the browser extension works now as a mediator to facilitate the web application to communicate with the mail servers. Then user don’t have to choose the suitable extension/app according to the browser he using. He need only to open up the web application. Then according to the browser, web application will automatically download and install the suitable extension/app.

Since this is a proof of concept, the app itself only supports mail fetching part. App also supports sending mail part though it is not implemented in the app user interface, but the internal functionality exist. So this product has lot more works to be done to make it usable.

But here we propose a more broad and general solution to this. Even this has mentioned a “client for browsers”, it actually doesn’t work on a browser. It runs as a Chrome app. Chrome app is not run in the browser, it runs in a isolated sandbox environment from the browser, as a fully-fledged application. Our solution implements a Chrome app to make a TCP connection to mail servers, but that app runs as a background application. It doesn’t has any user interface. The web application makes a connection with the app to communicate to the servers and all the data are visualize in the web application user interface.

Browser Mail has no any plans to support offline use. If user has no access to internet, then user can’t read, write emails. But our solution we have removed this barrier by storing emails within the browser database which proposed by HTML5 specifications. Web application will also cached in the browser using application cache API.

2.3. Socket-hub mail client

Sockethub is a polyglot (speaking many different protocols and APIs) messaging service for social networking sites and other interactive messaging applications [6]. Sockethub is developed to support unhosted web applications.

Not only does it assist unhosted and noBackend[5] web application developers by providing server-independent, server-side functionality, but it also can be used as a back-end tool (invisible to the user) for many different applications, large and small solution differs from it. What are the pros and cons.

As an example application, Sockethub has developed an unhosted email client [7]. It does very basic email functionalities such as sending and retrieving mails. What the sockethub server does is it communicate with mail servers with standard SMTP/IMAP protocols. Sockethub to client communication is done via WebSocket protocol.

After analyzing Sockethub architecture, we decided that this architecture doesn't preserve the unhosted web application architecture. Because user has to go through a 3rd party server which transform SMTP/IMAP protocols into WebSocket. The Unhosted nature prevails if the browser directly communicate with the backend. Backend should be swappable and the application logic should totally runs on the browser. Introducing a 3rd party server to facilitate the message transforming will break this nature. This key point led us to disqualify Sockethub server for using our unhosted email client.

Apart from that, user has to pass their credentials to the Sockethub server to authenticate with the mail servers. The Sockethub server will handle the authentication process with the mail servers. Storing and handling the user credentials could be very questionable and users will be not comfortable with sending their email credentials to a 3rd party server, even we hosted a Sockethub server locally. It introduce a new security threat.

But in our solution we directly communicate and authenticate with the mail servers, through the browser. Even we use an extension to communicate with the mail servers, it will not store any data in it. It runs independently within the users' computer, within the browser. So user don't have to worry about security issues and using a browser extension to facilitate the communication is quite acceptable.

2.4. Gmail Offline

Gmail offline is a Google Chrome app released by Google which can lets the users to read, write and manage emails when they don't have an internet connection [8]. Users can download this app from the Chrome app store [9] and install on Google Chrome browser. This app runs as a background app that means app itself doesn't has a user interface. User will redirect to a web application and this web applications contains all the user interfaces required.

After first start-up, Gmail Offline will automatically synchronize messages and stored them in the WebSQL browser database. When user don’t have internet all the emails can retrieved from the WebSql database, and the web application will be available through app cache.

Web Sql API is a browser database to store data that can be queried by SQL. The W3C Web Applications Working Group deprecated WebSql, stating that its specifications could not move forward to become a W3C Recommendation [10]. In our solution we propose indexdb as the browser database. It is the alternative for the WebSql. IndexedDB is a transactional database system, simply like a SQL-based RDBMS. But SQL-based RDBMS uses tables with fixed columns, IndexedDB is a JavaScript-based object-oriented database. All the objects stored in the indexeddb can indexed with a key. Because of this key indexing, indexdb performs efficient searching over values [11].

Since this is a Chrome app, this product has been also limited only to Chrome browsers. Gmail offline don’t have Firefox extension or any other extensions. But in our solution, what user all have to do is simply access the web application and, web application itself will decide which browser extension should be downloaded and installed, if it is not already installed. In the first release our solution will be containing both a chrome app and a Firefox extension.

Again this Offline Gmail web application and Chrome app only supports Gmail that means we can’t connect to any other email service system using this applications. We remove this limitation applying the unhosted web application. It makes the backend of the application swappable. Our proposed web application doesn’t tied up with any particular email service provider and this has given the user to connect to any email service as his/her wish.

In the security point of view, Gmail offline app have to authenticate the user with Gmail SMTP/IMAP servers when it works in online mode. But when user works in offline mode, no additional authentication required and users can access its mails directly from the web application. Google has stressed out that user should not install gmail offline application unless it is a personal device and no other persons has access to it [12]. Because any user who has logged into his user account can go to the browser and open the application, access the mails or directly access the indexeddb located in the browser.

Our solution also we didn’t try to add additional authentication process to the email client. Because today most of the people uses personal devices. So as long a person logged in to a device, we can assume he is the real owner of that device.

2.5. Firefox SimpleMail extension

SimpleMail[13] is a mail client run on Firefox browser as an extension. It is somewhere middle to the web-mail client and Desktop mail client application. Simple mail client is implement in such a way that it able to working in a browser tab same as HTML application. SimpleMail is a lightweight Firefox extension but it provide capability to working with number of mail accounts. Its main function is to provide off-line email function with the ease of email activities.

It support both IMAP and POP protocols to fetch the emails and SMTP to send the mail. It support all the security mechanism to communicate with mail servers including SMTP authentication (StartTLS)[14]. It maintain a contact list to support automatically completion, users convenient and remember the list of contact have used. As a glance it have the basic email features with offline functions like attachment, compose, reading, Folder operations etc plus optional functions.

SimpleMail is not an unhosted application but it is an offline mail client for Firefox browser. But it is the good sample to start implement unhosted mail client.

2.5.1. Problems in the SimpleMail

SimpleMail is not an unhosted web application. As describe earlier it is an extension to the Firefox. It implemented based on JavaScript as main logic implementation, CSS and XUL [15] for GUI implementation, style and structure it. It is implemented as an extension to the Mozilla Firefox. Thus to use it users have to installed the Firefox into the system.

2.5.2. Analyzing SimpleMail

SimpleMail is bit similar to the desktop mail client like thunderbird. Only different is, it work on top of the Firefox. It let user to add several account to the application. Then users can use mail accounts simultaneously. But it not provide account authorization when user start the application. That is user provide authentication details, only at the account adding time. After that no need to re-login to the account. But this is opposite to webmail client. It has login to the account when user need to use the mail service. If unhosted webmail client it should be able to connect to any mail services and logging out after the service is taken.

It is more Firefox dependent application. That is most of the javascript functions, Database service (Components.interfaces.mozIStorageService), network service, and are firefox dependent. Since firefox provide very useful and powerful API for extension development, SImpleMail was easy to implement on the Firefox browser. But It not easy to implement on other browsers such as chrome, Internet Explorer, Safari, Android browser etc. Those browser use separate their own extension development structure with different browser API and services. Even different browser has their own GUI implementation technologies rather XUL.

Our product is unhosted web application and do not couple with any server side script such as PHP. Thus implementation is based on HTML5, JavaScript and CSS only. Unlike the SimpleMail it has no XUL based GUI. Complete GUI of the Client is implemented in HTML5 and JavaScript is used to implement all other logical functions. Thus it is not dependent on the browser and platform.

Unlike the SimpleMail, Our product is not a similar to the desktop mail client or extension. It is web application with collection of HTML and javascript files. It is similar to the function of Gmail, Hotmail and Yahoo and not like thunderbird or Outlook. It provide User authentication in the Web application itself and SMTP/IMAP server authentication separately. User can login and logout when service is taking and finished. But possible to use any mail service at a time unlike the Gmail, Outlook or yahoo webmail client.

This product is not dependent for a particular browser. It not used Browser dependent API like firefox Components.interfaces.mozIStorageService inside the web application. It has used identified common service, technologies and function that can be used in most of the browsers. It simply used HTML5 as user interface without specialise in UI like XUL. Also it use indexDB for store the mail that used for offline activities rather service used in the SimapleMail similar to mozilla mozIStorageService.

Even it used common technologies and services there was few limited services that are depend on the Browser. Opening TCP connection to the IMAP and SMTP servers was heavily dependent on the Browser APIs it provided. Thus this product contain the two main part. One for Browser dependent and other is browser independent part. SimpleMail is completely browser dependent Application (firefox extension). But the same architecture used in our product to implement browser dependent part for Firefox browsers. When Application run, it find the Browser compatible Browser dependent part to open a TCP connection. Unlike the SimpleMail our product will open TCP connection using related to browser dependent services and APIs. SimpleMail not contain such a two parts or communication among both components. But our product contain the two part and those are communicate each other to send and receive the remote data through TCP connection. Thus it consider another communication mechanism such as custom JavaScript/HTML event or browser based mechanism which may be dependent on the browser as well.

There was no much suitable JavaScript IMAP/SMTP APIs for IMAP/SMTP operations. SimpleMail has used their own components to generate IMAP/SMTP command. Our product was same as SimpleMall and has used custom IMAP/SMTP command sender and receiving component to communicate with the server by passing messages. Since the JavaScript has asynchronous function execution and SMTP/IMAP communication also asynchronous, SimpleMail has used regular expression based message identification system to synchronize the response with the request. The same technique has used in our product since it has some advantages, simplicity and eases of implementation.

SimpleMail has defined the scenario based IMAP/SMTP server communication to define the unit of tasks. The same approach has used in our implementation which enable to gather sequence of request for a one task and putting it to same queues and execute.

2.6. Thunderbird Mail client

Mozilla thunderbird [16] is well known open source desktop mail client. Which is cross platform application and work with same environment Mozilla Firefox works. Thunderbird Provide more facilities and features than SimpleMail and it can work as a standard Software unlike the SimpleMail. Because Simplemail work on top of the Firefox browser. Thunderbird provide adding multiple Email accounts and provide offline activities as well.

2.6.1. Analyzing Thunderbird

ThunderBird need to set up before it use. Like basic software installed on the operating system, Thunderbird also have to install and configure. It’s about 30MB in size and it take time to users to use it. Thus users cannot use Thunderbird for a quick email services. After the Thunderbird installation users can't login to the prefered email account unless it configured. That is user should be configured appropriate mail server (SMTP/IMAP) addresses, ports and security details. Thus users should have some knowledge about their email service providers configurations details. As a solution for this issues we were able to give quick installation and configuration features to our unhosted mail client.

Thu unhosted mail client contain the two main parts. One is Browser dependent and independent part. Independent part is the HTML and JavaScript web application. And it should be accessible to devices through internet or browser cache location. Then the Browser dependent part download to the device according to the browser compatibility and it is different from browser to browser and may be to the platforms as well. The browser dependent component contain the extension or browser application part. After the extension is download it will install to the browser. Thus, user no need to spend much time to use the mail client. Unlike thunderbird this application size comparably very small. Thus in several seconds user’s browser ready for the application. With unhosted webmail client it have reduced the installation size and time when compare with thunderbird.

Desktop Email client that should be configure before they used according to the mail account. For that ThunderBird provide automatic configuration resolve mechanism [17]. For that mozilla maintain remote database called mozilla ISP database [18]. When the user enter the username and password it send query to the ISP database to search appropriate SMTP and IMAP servers.

Mozilla ISP database can use for any client according to the ISP database information. It still maintaining database and update for new configuration. Then in future it will be able to cover over 50% of configurations details. Thus it was power full and helpful to optimize the client configuration efficiency, we expected to integrate this client with mozilla ISP database. Since any organizations it is possible to update their server details with ISP, and the server information possible to fetch as simple XML format, it will easy to use such feature in our unhosted mail client.

With these solutions unhosted mail client can get optimized setup time and installation time. Thus even for a quick mail service this may help. It is easy to install, setup and easy to access and doesn't matter the platform or the browser.

Thunderbird use folder based hierarchical representation [19] to represent the messages. In a folder it keep the messages with same hierarchical level. In the message content it keep the headers plus message body with the attachment. In thunderbird it store this in two physical location. And every email account contain the root directory and subdirectory according to the mail account directory structure. In each Directory it keep the file to store the content of that message.

This file in the directories are consider as the thunderbird database. Database is composed of operating system related files and Thunderbird use two type files. mbox is the common and simple file to store the messages. It also used in the mail servers to keep the email content as well. In the mbox it store the list of messages in the same hierarchical directory. mbox is the plain text file and save messages one after the other. It is one of the best way to keep message because it not waste storage and very important if it is device with the limited storage. Since thunderbird installed on client computers, mbox doesn't take much storage capacity compare to the total storage. Other file type is msf file and it is in mork format. In that case message are indexes to provide fast access to the messages.

When it is web application it can't use or access the file system, since it is prohibited and block by the browser. Thus most suitable solution was to implement a database or file structure inside the web browser. Thus we use the indexDB since it is possible to create Databases on most of the browsers and it is new version of browser database. Application have two databases and one for keep the configuration and other web application metadata. Other one used to store the fetched mail and sent mail. It create one database for each account and one database contain number of table related to the particular mail boxes. If there is more than one hierarchical folders we modify the table with its absolute path. In the tables, it will store the mail in each rows.

For logging to the databases it need a username and password. Thus simply it use the account username and password. Before it used, it hashed the data as appropriate and store in the metadata database. Unlike the thunderbird it not store the complete or same mail as in the remote mailboxes in the databases. It has defined the most appropriate attribute in the mail content and attributes, that visible to the users from the interfaces. Then only that details are stored in the databases.

The synchronization module interact with the database to synchronize the browser database with remote mail boxes. Periodically it refresh and compare the mailboxes to update the browser database. In thunderbird each and every mail account have incomingserver entity [20]. In our approach it allocate own two instance of connection for a one account. One instance used for SMTP and one for an IMAP. This was help because then it is possible to simply manage the response and request. Because JavaScript and server responses are asynchronous, this individual connection given an efficient interaction with servers. It identify each connections by an id. That id is used to communicate with browser dependent part to communicate with server. When that communicate happen connection id is always used to identify the mail account with respect to the connections.

2.7. Unhosted email client [21]

2.7.1. The context

As explained earlier, unhosted email client is an email client which is built on top of unhosted web application architecture [21]. To simplify the norm, it can be thought as Gmail application entirely running on a browser built using JavaScript. It is an initiative of unhosted.org. Complying to unhosted architecture, the email client has to an intermediate server running remotely to relay the requests. As web apps cannot open socket connections to random end points, SocketHub[6] has been used for relaying. Even newly introduced web sockets cannot open SMTP/IMAP connection to servers. As mentioned here, it facilitates clients to connect to few web services. Here, sockethub can handle TCP communication to IMAP/SMTP servers. It helps to transform websockets connection from web apps to IMAP/SMTP connection to servers [6]. The relaying happens in reverse way as well ensuring two direction communication.

This email client can connect to another backend where users can store received/sent emails. “remoteStorage.js”[22] has been used to run the own backend server. The remoteStorage is an open protocol for per-user storage. Anyone can build their own backend services on top of remoteStorage. Using remotestorage one would be able to migrate data to another server running remotestorage. Actual picture here is receiving mails from intended email server and storing them into our own storage so that to create a norm of owning our own data. Storing the mails received has not been done. When looking at the core mechanism here is, to send mail the the application sends basic attribute components for an email message encapsulated into an object to sockethub. Then sockethub receives the object and extracts elements and forms the smtp message. Once the smtp message is formed, it is sent to intended smtp servers to further relaying. On the other side, to receive mails, a request is made to sockethub then the sockethub retrieves mails and forwards it to the remotestorage.

Using the system one can send email with recipient, sender, text body and subject. Cc and Bcc facilities are also available. Additionally fundamental level of message threading is also supported. Inbox search can also be done using “grep” command on the file system of the VPS[21]. But this is not efficient as other existing inbox search facilities offered by mail clients.

2.8. Relevancy analysis

As pointed out above, in the overview architecture, the developed unhosted email system has an intermediate relaying server to transform the tcp connection [21]. More or less the intermediate server plays a role as a proxy to the mail servers. So the system is tied up with another server which hosts the sockethub module. But the actual intended architecture does not require an intermediate server to relay the connection. Unlike this system, the application requires to connect to mail servers directly from browsers itself. To be more precise, the functionalities of sockethub needs to be injected into the application itself and it should run within the browser's context. The main challenges of opening raw tcp socket connection from browsers is not addressed here. This is handed over to sockethub. But it is obvious and simply possible to open the above connections to a random end points from a server. It is possible for the above system as it hosts the sockethub module in a remote server. The addressable issue here is relaying through a third party server, as the data is high sensitive. Normally, if one tries get services from sockethub, it has to be planted on a server in the Internet. So it is not possible to convince email system users about the security and confidentiality. Only way to trust the sockethub is to run it in the trusted local network. Then we can have end to end encryption. In the proposed system, all the connections are secured as the browsers open secured connection (TLS/SSL) to email servers. And as the system connects to multiple email servers, it not possible to have separate common encryption mechanism.

Sockethub module is still unstable supporting connections to one or two main mail services. So the unhosted email client can only be able to connect to those email servers. As this system highly relies on sockethub, it cannot connect any random email servers. It needs to wait until the sockethub starts supporting the facility. And it is unlikely that sockethub community would provide the facility of relaying to email servers of custom/personal domains. If solution is needed, sockethub module needs to be customized as it is open source. But still it is not possible or beneficial to spend effort to customize it. Unlike this, the proposed solution can connect to any email servers, providing information and credentials of email accounts from any email servers in the application itself. Any issues in the sockethub can affect the system functionalities starting from opening the connection to other mail transferring functionalities. And the system would be down if anything goes wrong in the server in which the sockethub module runs, irrespective any issues of email servers. Availability of the system could be reduced by having sockethub in the middle.

Another main focus is on storing retrieved mails into users own servers which runs remotestorage.js. This is an element of unhosted web application architecture. Then again, one needs to run the remotestorage.js in a secured server. It can be localhost or any other host in the Internet. How would one rely on third party server to store email messages which are highly confidential data. But the only acceptable reason is that the user can have own backup, in case if the email service is stopped by the vendor or any other reason. The proposed solution stores mails in the browsers storage. But lacks to backup large pile of email messages considering confidentiality and security. Still the browser storage plays a role as small backup of user's recent emails.

As the norm of per-user backend is emphasized, the unhosted email client from unhosted.org community does not dedicate much on email server swappable feature. When considering the client component, it seems to have no facility to cache the resources to in the browser. Thus, it proves no availability when not connected to internet. After all, the system is still in an unstable position lacking lots of important features of an email system.

2.9. Rest based offline email system [23]

2.9.1. The context

The system is based on REST and stores emails and meta data in browser's storage [23]. It is for web based email systems. And it requires no plug-ins to be installed in the browsers. It mainly uses HTML5 technologies to utilize the offline features by accessing browser's storage. The system tries implement its own web mail servers as traditional email servers have the IMAP/SMTP interfaces[23]. The interface is to support REST. Then the communication can be directly made from the browsers.

The system has two main components as client side and server side. Server side has main three components as database for storing mails, reception for incoming email messages and REST web services. REST services are used instead of IMAP. For each IMAP command there is a respective service call. Email clients can use the service call to retrieve and send emails. There is another module to handle the sync with offline features. Using REST, the clients can retrieve a particular email or its part. This is way of reducing network traffic.[23]

Client side has three modules consisting HTML5 local storage, Offline HTML5 web user interface, synchronization module. Here the client side database is key-value database stored inside browser's storage. The web client is structured in two main components as graphical user interface using HTML5, css and application logic using AJAX and javascript.[23]

2.9.2. Relevancy analysis

As the system implements two components as client and server side, client side reflects many features that the proposed system has. Main difference is that this system uses REST based services and is attached to its own email servers. This is highly reduces the flexibility. And it does not open imap/smtp connection (raw tcp connection) from browsers. Instead the system has implemented rest services. Messages are passed using rest services. As the system has its own email servers, it can not connect any random email servers like the proposed system.

Even though the system has the rest api as an overlay for imap/smtp interface, the system has introduced service calls to efficiently handle the network traffic. This has a lot of advantages compared to imap connections. Because needed information can be retrieved from the server unlike retrieving everything and parsing it to get relevant data.

As the system provides REST API, using URL to access resources, anyone can create their own email client to use this server side components, ultimately making an email system. As REST APIs are easy to understand unlike IMAP command, it is much easy to implement our own client. As system uses HTTP connection, the reliability of the communication falls on the http protocol standards.

In the client side, the system partially coincides with the proposed system as it supports offline functionalities. Especially offline features are in both system uses html5 capabilities. As the system uses offline features by storing emails it needs to have an effective sync engine to update the unsynced data.

2.10. Gmail API[24]

2.10.1. The context

The gmail email platform provides flexible Restful APIs to use its emailing facilities. APIs are for sending messages, receiving messages, label handling and searching messages in inbox. Anyone with the help most of the programming languages can utilize the APIs to handle emailing facilities into the application. By including the client library, one gmail user can these facilities using his/her credentials. This client libraries are available in different languages. Application developer can download relevant one and use. Well-structured documentation is also provided by the gmail community. As it supports for many different languages, mobile application can also get use from it.[24]

2.11. Relevancy analysis

The whole idea here is to including the client library to simply send and receive mails via our application. But the application still can extend a dedicated email client using Gmail APIs. This is more or less like developing your own graphical interface for already existing gmail application. More importantly all the API calls are restful. That requires only the http connection. So its possibility of building an application that supports most browsers and mobile environments is high. The concern here is that the APIs are only dedicated to gmail services. The application which uses the APIs can connect only to gmail services, not the other email services. This is where it contradicts with the proposed solution. Even though it facilitates the http accessibility to connect to gmail servers rather than using imap connection, it is not backend swappable to connect to other email servers. It is not a common API to have access for other servers. And making this APIs flexible is what the proposed solution needs, ending up it as common interface for imap/smtp connection.

2.12. Browser plugins/apps[25], [26]

Plugins/extensions are used by most of the browsers to extend the browsers capabilities. Browser vendors provide browser specific APIs to utilize browsers' capabilities. Extensions use this APIs to implement extra features to the browsers. Extension may provide facilities to web pages. Extension/plugin can have their own window as well. Further browser application can run on the browsers having its own window interface. But ultimately they also use browser APIs to implement the functionalities. This architectural norm can be thought as an application running on an operating system. Browser can be thought as an operating system.

Because of security concerns, any web page can not open raw tcp connections to a random server running from a browser. But browsers provide API to open connections to a server. Other issue is a web page running on a browser can not directly uses browsers APIs. This is where the extensions play a vital role. To open raw tcp connection, web pages need to pass messages to extensions/browser apps. Sequentially the extension can open tcp connections to the intended server. So here extension or browser apps can communicate with web pages. As the the proposed solution needs imap connection to email servers, the solution needs to have extensions as intermediate server between unhosted application and random email servers.

In firefox browser, extension used to open the imap connection[25]. The enabled extension runs in the background of browser listening to the events. Relevant web page fires an event with messages to be passed to the extension. Extension catches the event and proceed the task assigned in the event listener. Unlike firefox, chrome browser extensions doesn't have permissions to create tcp sockets[26]. But chrome application have the permissions to create sockets and open tcp connections[26]. Then, message passing mechanism is abstractly the above. Both of the browsers provide powerful APIs to create sockets , open connections and send imap, smtp commands.

3. References

[1]Jong ,Michiel de,” unhosted web apps”, unhosted web apps. [Online]. Available: https://unhosted.org/

[2] PJoe, “Linked Data Platform for Web Applications” , Available: http://dig.csail.mit.edu/2014/Papers/presbrey/thesis.pdf

[3]Ram, “HiddenTao – Web hacker, Javascript enthusiast”, Javascript IMAP email client for browsers. . [Online]. Available: http://www.hiddentao.com/archives/2013/08/15/javascript-imap-email-client-for-browsers/

[4] Jong ,Michiel de, “Build apps like there is noBackend!”, Bring your own Backend!. [Online]. Available: http://nobackend.org/2013/11/cross-origin-backend.html

[5] “Sockethub”, Sockethub. . [Online]. Available: view-source:http://sockethub.org/
“Sockethub Examples”, Sockethub Examples. !. [Online]. Available: https://silverbucket.net/sockethub/examples/#/

[6] “Gmail Offline: Get started - Gmail Help”, Gmail Offline: Get started. [Online]. Available: https://support.google.com/mail/answer/1306847?hl=en

[7]“Gmail Offline: Sync &amp; storage - Gmail Help”, Gmail Offline: Sync & storage. [Online]. Available: https://support.google.com/mail/answer/1306849?hl=en

[8] “Indexdb” Indexdb Intro[Online]. Available: http://www.w3.org/TR/IndexedDB/

[9] “Indexdb API” Indexdb API Intro. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

[10] “Indexdb API” Key concepts and usage. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

[11] “noBackend”, Go noBackend today!. [Online]. Available: http://nobackend.org/solutions.html

[12]“Getting Started with noBackend”, Getting Started with noBackend [Online]. Available: http://nobackend.org/blog.html

[13]“Simple Mail :: Add-ons for Firefox,” Simple Mail :: Add-ons for Firefox. [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/simple-mail/.

[14]C. Newman, “Using TLS with IMAP, POP3 and ACAP,” RFC 2595 - Using TLS with IMAP, POP3 and ACAP, Jun-1999. [Online]. Available: https://tools.ietf.org/html/rfc2595.

[15]“XUL - Mozilla | MDN,” XUL - Mozilla | MDN, 14-Apr-2014. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL.

[16]Sheppy, “Mozilla Thunderbird - Features,” Mozilla Thunderbird - Features. [Online]. Available: https://www.mozilla.org/en-US/thunderbird/features/.

[17]BenB, “Autoconfiguration in Thunderbird - Mozilla | MDN,” Autoconfiguration in Thunderbird - Mozilla | MDN, 14-Feb-2014. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Thunderbird/Autoconfiguration.

[18]“ISP Database | Thunderbird Help,” ISP Database | Thunderbird Help. [Online]. Available: https://support.mozilla.org/en-US/kb/isp-database.

[19]“Major Thunderbird Entities,” User:Emre/tb/architecture/diagrams - MozillaWiki, May-2008. [Online]. Available: https://wiki.mozilla.org/User:Emre/tb/architecture/diagrams.

[20]“Interface,” User:Emre/tb/architecture/diagrams/messageincomingserver - MozillaWiki, Apr-2008. [Online]. Available: https://wiki.mozilla.org/User:Emre/tb/architecture/diagrams/messageincomingserver.

[21] Michiel B. de Jong, “Sending and receiving email from unhosted web apps,” [Online] Available: https://unhosted.org/adventures/9/Sending-and-receiving-email-from-unhosted-web-apps.html

[22] RemoteStorage community, “remoteStorage-an open protocol for per user storage,” [Online] Available: http://remotestorage.io/

[23] Gihan Dias, Mithila Karunarathna, Madhuka Udantha, Ishara Gunathilake, Shalika

Pathirathna and Tharidu Rathnayake. *REST-based Offline e-Mail System*. Proceedings of the Asia-Pacific Advanced Network 2012.

[24] Google developer community, “Gmail API,” [Online] Available: https://developers.google.com/gmail/api/

[25] Mozilla Developer Network, “Add-on SDK” [Online] Available:
“https://developer.mozilla.org/en-US/Add-ons/SDK

[26] Google chrome developer community, “Message Passing” [Online] Available:
https://developer.chrome.com/extensions/messaging