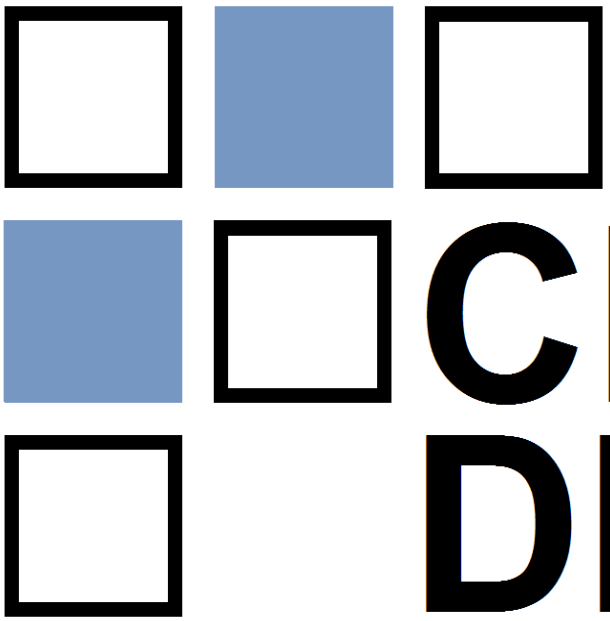


A cloud based visual bookmarking service

TECHNICAL SPECIFICATION

Shane McQuillan – 58600724
Dr. Gareth Jones
http://student.computing.dcu.ie

Table of Contents

1. Introduction

1.1 Overview

Cloud Dial is a web application that allows users to store, manage, retrieve and share their bookmarks anywhere, any time, straight from their browser. This system aims to be central to a users' web browsing. It should be the first thing they see when they open their browser, a new tab, or new window. Doing so they can seamlessly access their favourite sites and collected bookmarks, greatly improving productivity. In Cloud Dial each bookmark will be identifiable by an image. This will make it visually appealing and extremely practical to use in this way, especially on mobile devices.

Identifying bookmarks further will be made possible by means of textual tags, content summarisation, and categorisation into groups and sub-groups; all of which will be suggested by Cloud Dial. This rich variety of information will ensure efficient browsing, and effective bookmark retrieval.

It is hoped this system will be used by all web browser users, but more specifically, by those who bookmark regularly. An ideal user bookmarks on several different devices, bookmarking pages of various topics. For people who bookmark a lot, and need to retrieve specific bookmarks on many devices, Cloud Dial is the perfect solution.

1.2 Motivation

People visit the same group of web pages every single day. They do this on their laptop at home, their PC at work, and on their smart phones and tablets while on the move. This means either bookmarking these pages on each device individually, or laboriously entering the URL or search query for every visit of these pages.

In addition, many people bookmark pages that they don't necessarily visit a lot but need a record of nonetheless. However it is regularly the case that a page is bookmarked on one device but not on another, which all to often leads to people being unable to find desired pages.

Keeping bookmarks in sync between all devices can be extremely tedious. Operating systems and browsers offer solutions to this, but this leaves you tied to the chosen software. Cloud Dial aims to rectify this.

Products similar to Cloud Dial exist. However in order to avail of all the functionalities Cloud Dial provides, it is necessary to use more than one of these products. Cloud Dial combines all of these services in a simplified manner, making life a little easier.

1.3 Glossary

Note: Any words emboldened and italicised in this document are explained in the glossary.

Stop Word: Stop words are words which are filtered out prior to, or after, processing of natural language data.

DOM: The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents.

MVC: Model–View–Controller is a design pattern for computer user interfaces that divides an application into three areas of responsibility:

1.the Model: the domain objects or data structures that represent the application's state.

2.the View, which observes the state and generates output to the users.

3.the Controller, which translates user input into operations on the model.

JSON: JSON or JavaScript Object Notation, is a lightweight text-based open standard designed for human-readable data interchange.

NoSQL: In computing, NoSQL is a class of database management system identified by its non-adherence to the widely used relational database management system (RDBMS) model.

Middleware: In its most general sense, middleware is computer software that provides services to software applications beyond those available from the operating system. Middleware can be described as "software glue".

Facebook Open Graph: The Open Graph protocol enables developers to integrate their pages into the social graph. These pages gain the functionality of other graph objects including profile links and stream updates for connected users.

REST: Representational state transfer is a style of software architecture for distributed systems such as the World Wide Web.

Lucene: Apache Lucene(TM) is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

Asynchronicity: Asynchronous events are those occurring independently of the main program flow. Asynchronous actions are actions executed in a non-blocking scheme, allowing the main program flow to continue processing.

2 Research

It was necessary to research several areas before implementing certain features of Cloud Dial. As part of this research I attempted to identify different approaches that might work well for implementing these functionalities. I detail the various approaches below, with how applicable they are to Cloud Dial. The implementation section details the chosen approaches.

2.1 Tagging

A tag is a keyword used to describe an article or website. Tagging is useful in the context of bookmarking because it allows users the flexibility of classifying their collections of items in the ways that they find useful. This greatly aids categorisation, as well as later retrieval of content.

However tagging content is often viewed as laborious. As such this step is often skipped, or used poorly. In order to help with process Cloud Dial will suggest tags. Here is some of the methods that can be utilised for tag suggestion, along with how applicable they are to Cloud Dial -

- Term Frequencies

If a thorough stop word list is used, words with high term frequencies are likely to be appropriate keywords for a document.

This is a simple solution that could work well with Cloud Dial. However it will be unable to suggest keywords that are not present in the content.

- Record Fields

Certain fields may contain keywords that are more appropriate for describing content. The title field in particular will usually contain the most relevant keywords. This can exploited to improve tag suggestion.

If a title field exists Cloud Dial could undoubtedly use this to give greater weight to title terms.

- Keyword Meta Tags

Many pages on the WWW have keyword meta tags to aid search engines. These are usually hidden from users, but could be extracted and suggested as tags.

This could work well for Cloud Dial, however it is often the case that the home page of a website alone contains these meta tags.

2.2 Representative Images

Here is a link, what image best represents it? This is a question that has not been considered by many. There's no denying it, images are more appealing to the eye than text. People engage with things that catch the eye. Why then has no one solved this problem? Here is some possible solutions, along with their relevance to Cloud Dial -

- Content Image Extraction

Images contained with the content are likely to be relevant to the content. Extract image candidates from content DOM, and rank these according to meta data.

This is a good solution to the problem, but some pages have no images, and others have a massive variety of images that are irrelevant.

- External relevant images

Well formed queries to image search engines may return high quality images. Some possible queries are -

- The title of the page

- The domain name of the page. eg. 'facebook' for 'http://www.facebook.com'

- A concatenation of tags relevant to the page (but how many?).

A combination of all these options may also perform effectively.

- Screenshot

A simple screen shot of the page is undoubtedly relevant. However the result will not always be visually appealing.

2.3 Summarisation

A summary is a condensed derivative of a source text. i.e. content reduction through selection or generalisation on what is important in the source.

Document and content summarisation is a long established area of research. It is affected by numerous factors, including the form of the source text, the intended purpose of the summary, and the degree of coverage of the input and the format, to name but a few concerns. Because the forms and needs of summaries are so varied, it seems unlikely that a single general technique for automatic summarisation is possible, however for use in Cloud Dial the summarisation technique is defined, so we can apply general summarisation techniques. Summarisation in Cloud Dial will be used to suggest content descriptions to users. It is hoped a high quality summary will sufficiently describe the content of an added bookmark for a user.

There are numerous methods used to automatically summarise content. Below I detail several of these approaches and their relevance to summarisation in Cloud Dial -

- Luhn's Keyword Cluster Method

Luhn concluded that the frequency of a word occurrence in a document and its relative position determines its significance in that article.

```
sentence score = #(significant words in sentence)^2/#(words in sentence)
```

For use by Cloud Dial significant words are those with the highest term frequencies. These are matched against a detailed stop words list. How many of these highest terms to use was determined experimentally.

- Title Terms Frequency Method

The title of an article often reveals the major subject of that document.

```
sentence score = #(title terms in sentence)/#(terms in title)
```

This is a perfect approach for Cloud Dial. However the possibility of no title existing needs to considered.

- Location/Header Method

The first and concluding sentences of a document often provide important information about the

content of the document. These sentences can be assigned a location score.

```
Sentence score = 1/#(sentences in the document)
```

Very applicable to Cloud Dial, however the percentage of sentences from the start and end to score need to be chosen with care.

- Query-Bias Method

Bias factor to score sentences containing query terms more highly.

This method is used effectively by search engines, however it is not applicable to Cloud Dial. Users may discover an added bookmark by querying a search engine. However gaining access to these query terms would not be possible in most cases.

- Description Meta Tag

Many pages on the WWW have description meta information to aid search engines. This is usually hidden from users, but could be extracted and suggested as a content summary.

This could work well for Cloud Dial, however it's often the case that the home page of a website alone contains this meta information.

2.4 References

[1] Hiemstra, D. “Using language models for information retrieval”:81-82

[2] Teller, S. "What's the best thumbnail for this page?"
http://www.zemanta.com/fruitblog/whats-the-best-thumbnail-for-this-page/

[3] Jones, G. Summarisation notes for CA437.

[4] Pal, S. “Summarization with Lucene”
http://sujitpal.blogspot.com/2009/02/summarization-with-lucene.html

3 Design

3.1 System Architecture

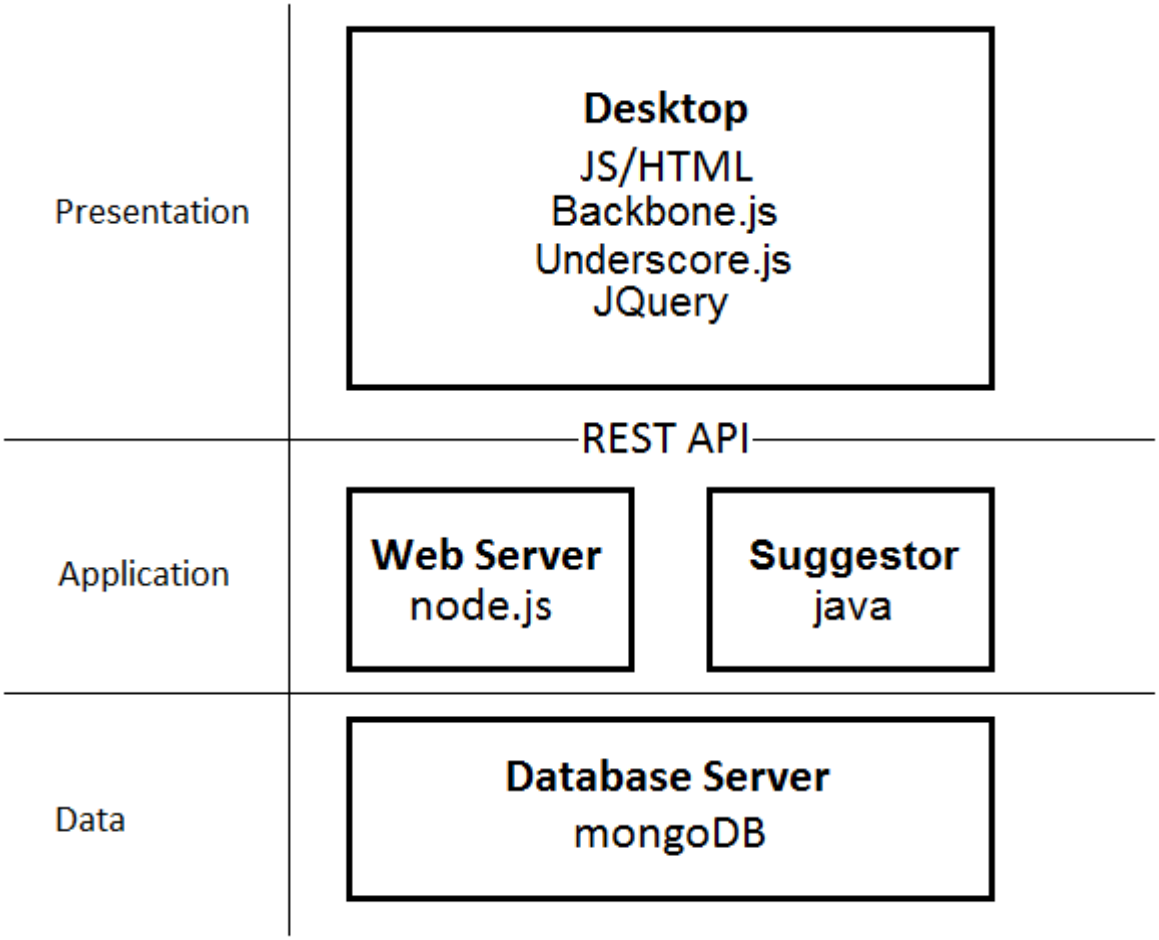


Fig 3.1

Fig 3.1 Is a high level representation of the architecture of Cloud Dial. It is 3-tier architecture.

3.1.1 Presentation tier

The presentation layer is concerned only with presentation. It is developed using HTML5, the MVC framework backbone.js, and JQuery for the interface logic. Server side templating engine Jade, and client side templating engine Underscore are also used.

3.1.2 Application tier

The application layer contains the logic of the application. It controls the application's functionality, and is central to its operation.

It contains the application server which is developed using node.js. Any computationally expensive tasks are developed using Java to aid peformance. The suggestions functionality uses Java primarily.

3.1.3 Data tier

The data layer contains the application's database. This is where all user and bookmark information is stored and retrieved.

Mongodb is used for storage. The reasons for this are outlined in the implementation section.

3.2 High Level Design

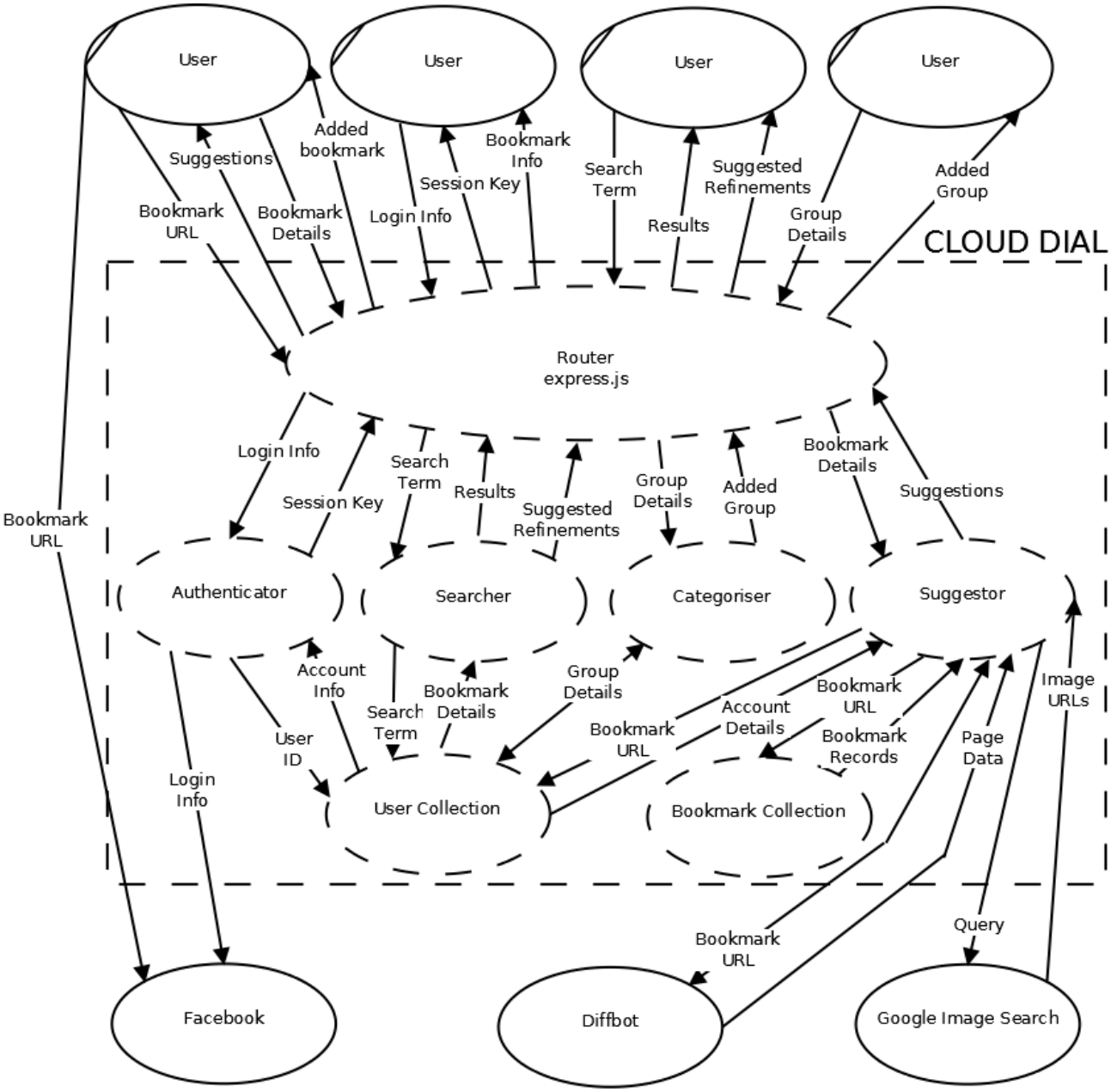


Fig 3.2

Fig 3.2 is a high level data flow diagram(DFD) of Cloud Dial. It shows its main components, both internally and externally.

3.2.1 User

This represents an active user of Cloud Dial. The DFD considers several user instances. These are external to Cloud Dial.

3.2.2 Router

This is central to all functionality in Cloud Dial. Requests are sent, and are routed by the Router to the appropriate component. This is internal to Cloud Dial.

3.2.3 Authenticator

The Authenticator is used by Cloud Dial to authenticate users on login. It uses either Facebook to do this, and if approved a session key is passed. This is internal to Cloud Dial.

3.2.4 Searcher

The searcher component is used for bookmark searching. To do this the component needs a query and a search space. This information will be routed to the Searcher component on the event of a search. This is internal to the system.

3.2.5 Suggester

The Suggester is responsible for making intelligent suggestions to a user. Namely tag suggestions, image suggestions, summary suggestions, and grouping suggestions. To do this the component needs the URL of a bookmark being added. Using this it can analyse the content and make suggestions based on underlying assumptions. This is an internal component.

3.2.6 Categoriser

The categoriser is used for grouping bookmarks. It deals with new group creation as well communicating group information. It is internal to Cloud Dial.

3.2.7 Facebook

Facebook is used for authentication. For this Cloud Dial must communicate user information with Facebook.

Users can also allow automatic updates to their Facebook account on addition of bookmarks. For this bookmark information must be provided. This is external to Cloud Dial.

3.2.8 Google Image Search

Google's Image Search API is used to suggest images to users. Cloud Dial provides a query to the service, and results are returned. It is external to Cloud Dial.

3.2.9 Diffbot

Diffbot is used to parse data from web pages. Cloud Dial makes a request to Diffbot with the page in question's URL. Diffbot returns all parsed data. It is external to Cloud Dial.

4 Implementation

4.1 Presentation

4.1.1 Interface Design

Cloud Dial's interface design is minimalist in order to avoid confusing users. Minimalism is achieved by reducing a design to only the most essential elements. This is used in other to keep the design attractive, but simple. Cloud Dial has many features which could easily become overwhelming for the user. Using minimalist design avoids this problem.

4.1.2 Interactivity

Several JavaScript libraries are used to make Cloud Dial interactive, most notable of which is Backbone.js. Backbone.js in a nutshell, is a framework which provides structure to your JavaScript and jQuery code. The problem is standard JavaScript libraries are great at what they do, but without realizing it you can build an entire application without any formal structure. Backbone solves this problem for Cloud Dial by providing an MVC structure to the presentation layer.

Backbone depends on library Underscore.js to work correctly. Underscore provides a client side templating solution which proved very useful for Cloud Dial. Templates allows you you to keep HTML out of your JavaScript code, and provide variables to chucks of HTML. These were used in Cloud Dial to model bookmarks, groups and tags.

Bookmark

```
< script type='text/template' id='bookmark-template'>
   <li class="dial">
      <a class="dial-image" href="<%= address %>">
         <% if (imgAddress) { %>
            <img src="<%= imgAddress %>"/>
         <% } %>
      </a>
      <h5><%= title %></h5>
      <div class="info-bookmark">?</div>
      <div class="del-bookmark">x</div>
      <div class="hidden tooltip">
         <img class="tooltip-arrow" src="../images/arrow.png"/>
         <div class="tooltip-content">
            <div class="semi-important popup-section"><%= title %></div>
            <div class="popup-section"><%= description %></div>
            <div><%= tags %></div>
         </div>
```

```
        </div>
      </li>
   </script>
```

Group

```
   <script type='text/template'  id='group-template'>
      <li>
         <a><%= name %></a>
      </li>
   </script>
```

Tag

```
   <script type='text/template' id='tag-template'>
      <li>
         <div class="tag">
            <div class="co">
               <span class="name"><%= tag %></span>
               <div class="d"></div>
               <span class="count"><%= amount %></span>
            </div>
         </div>
      </li>
   </script>
```

Cloud Dial is essentially a one page application, so it is unavoidable that it will become JavaScript heavy. Backbone alleviates this problem considerably.

4.2 Application Server

Cloud Dial's application server is developed using Node.js. The creators of Node describe it as a tool “for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.”

Due to the large amount of I/O in Cloud Dial, Node seems to be the perfect fit. Its non-blocking I/O model combined with JavaScript make it a great choice for wrapping other data sources such as databases or web services and exposing them via a JSON interface. The fact that programs in Node are also written in JavaScript means almost the entire code base for Cloud Dial is one language, which greatly eases code maintenance.

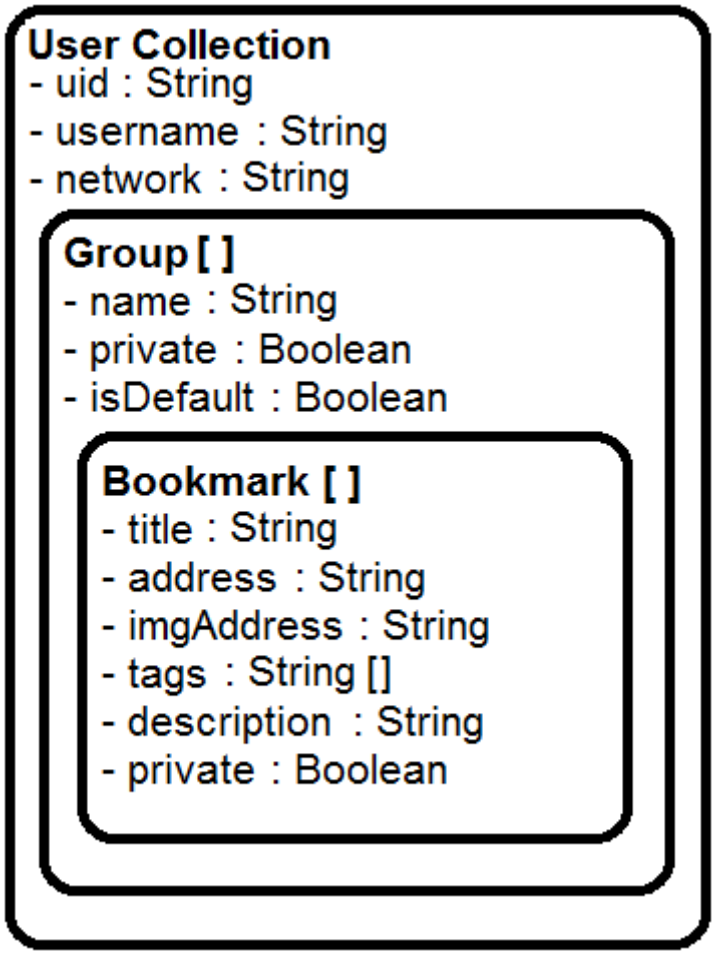
Like any tool Node does have weaknesses. It performs poorly for computationally expensive tasks, and as a result it was necessary to write parts of Cloud Dial in Java. More detail is provided on this in later sections.

4.3 Storage

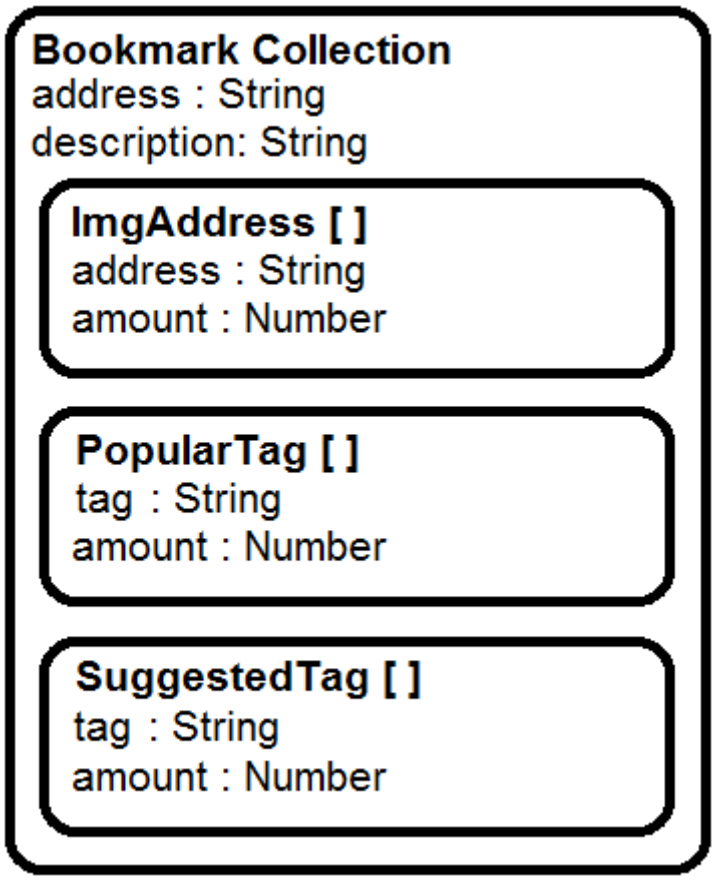
For storage Cloud Dial uses NoSQL database MongoDB. Being NoSQL, MongoDB is obviously quite different to traditional SQL databases. It describes itself as “scalable, [and] high-performance”, but it was primarily its ability to store multi-dimensional data structures that made it ideal for use in Cloud Dial. Embedding is pre-joining, which makes development much easier, and improves performance greatly.

4.3.1 User Collection

Note: Collections are analogous to tables in relational databases.



4.3.2 Bookmark Collection



Cloud Dial uses Mongoose, a MongoDB object modeling tool for Node, to interact with the database. The great thing about MongoDB is that you get JSON objects as query results. With such a JavaScript heavy app, getting JSON objects from the database makes the development process much easier. The fact that embedding can be used to provide structuce means it's often it is the case that the query results can be passed straight to the client without any processing.

4.4 Authorisation

For authentication Cloud Dial uses Everyauth, middleware developed for Express to authenticate and authorize users. Everyauth allows the addition of multiple login services, however for use in Cloud Dial, it uses Facebook alone. Everyauth allows to you to specify what permissions you want for a users' Facebook account. This was very useful for integrating with Open Graph (detailed later). The code below shows the settings used with Everyauth. Everyauth's methods are chainable.

```
everyauth.facebook
  .appId(config.fb_id)
  .appSecret(config.fb_secret)
  .scope('publish_actions')
```

```
.findOrCreateUser(function (session, accessToken, accessTokenExtra, profile) {
  var promise = this.Promise();
  user.findOrCreateByUidAndNetwork(profile, promise, function(){});
  return promise;
})
.moduleErrback(function(err){
  console.log('Error logging in');
})
.redirectPath('/');
```

When a user logs in for the first time they must give Cloud Dial permission to use their basic Facebook account information, as well as to post on their account page. If the user accepts this their information is stored in Cloud Dial's user collection. This information is retrieved for all future logins.

4.5 Categorisation

The ability to categorise bookmarks is very similar to the top level directory structure of most file systems. From your Cloud Dial account you have access to created groups, like the directories you have access to from a Unix root directory. These groups contain your bookmarks.

It should be clear in the User Collection database schema as well as the front end class diagram how groups fit in. MongoDB makes things very easy here. Because the data is already structured and stored as JSON, we can pass it straight from the database to Backbone's App class. The initialisation methods of the Group and Bookmark classes ensure the structure is consistent between the front and backend.

Further categorisation is possible by creating more groups. The necessary information can be passed to the server using the group function of Cloud Dial's REST API. The details of this are documented with the complete API.

4.6 Bookmark Addition

Logged in users can add bookmarks to their account. Users have the option of adding a bookmark from their account page, or using the Cloud Dial bookmarklet.

4.6.1 Bookmarklet

A bookmarklet is a bookmark containing executable JavaScript that can be used on any site. Cloud Dial's bookmarklet makes a call to Cloud Dial sending it the URL of the page. This presents the user with with a new window for bookmark addition populated with the page information, and suggestions from Cloud Dial.

```
javascript:
(
  function() {
    f = 'http://localhost:4444/savescreen?address='
```

```
        +encodeURIComponent(window.location.href);
    a = function() {
      if(
        window.open(f,
            '_blank',
            'location=yes,links=no,scrollbars=no,toolbar=no,width=550,height=550')
      ) location.href = f + 'jump=yes'
    };
    if(/Firefox/.test(navigator.userAgent)) {
      setTimeout(a,0)
    } else {
      a()
    }
  }
)()
```

When saved Backbone's save method is called on the created bookmark. This makes a call to Cloud Dial's REST API. The details of which are documented with the complete API.

Client Side:

```
    var bookmarkTitle = $('#bookmark-title').val();
    var bookmarkUrl = $('#bookmark-url').text();
    var bookmarkImageUrl = $('#bookmark-image-url').val();
    var allTags = $('#bookmark-tags').val().split(/, /);
    var desc = $('#bookmark-desc').val();
    var priv = $('#private-bookmark').is(':checked');
    var bkmrkInfo = {
          title: bookmarkTitle,
          address: bookmarkUrl,
          imgAddress: bookmarkImageUrl,
          tags: allTags,
          description: desc,
          private: priv};
    var bookmark_model = new Bookmark(bkmrkInfo);
    var group_name = $('#group-name :selected').text();
    bookmark_model.save({group: group_name});
```

Server Side:

```
exports.saveBookmark = function(req, res) {
```

```
    if(req.user) {
       var bkmrk = {
          'title': req.body.title,
          'address': req.body.address,
          'imgAddress': req.body.imgAddress,
          'tags': req.body.tags,
          'description': req.body.description,
          'private': req.body.private
       };
       user.addBookmark(req.user._doc.username, req.body.group,
                              bkmrk, function(err){});
       bookmark.addBookmark({'address': req.body.address, 'imgAddress':
                 req.body.imgAddress, 'tags': req.body.tags}, function(err){});
    }
};
```

4.6.2 Realtime

This feature was inspired by Apple's iCloud. When a user adds a bookmark it immediately appears on their account page on any logged in device. Socket.io by Guillermo Rauch is used for the implementation. Socket.io aims to make realtime apps possible in every browser and mobile device, blurring the differences between the different transport mechanisms.

On opening of a users Cloud Dial account a realtime connection is made with the application server. When a user adds a bookmark this connection is notified, and the results are sent to all sessions stored under that username.

Client Side

```
socket.emit('addition', {'group': name, 'bookmark': bkmrkInfo}, function(data){});
```

Server Side

```
var sessionConnections = {}
socket.on('sconnection', function (client, session) {
  var username = session.auth ?
                    session.auth.facebook.user.username |
                    session.auth.facebook.user.id : null;
  if(username) {
    var username = session.auth.facebook.user.username;
    if(!sessionConnections[username]) {
      sessionConnections[username] = new Array();
```

```
        sessionConnections[username].push(client);
      } else {
        sessionConnections[username].push(client);
      }
    }

    client.on('addition', function (data, fn) {
      var username = session.auth ?
                        session.auth.facebook.user.username |
                        session.auth.facebook.user.id : null;
      if(username) {
        sessionConnections[session.auth.facebook.user.username].forEach(function(sock)
{
          sock.emit('bookmark', data);
        });
      }
    });
  });
```

4.7 Bookmark Deletion

Logged in users can delete bookmarks from their account. Doing so invokes Backbone's destroy method which makes the appropriate delete call to Cloud Dial's REST API.

Client Side:

```
   bkmrk.destroy();
   var index = -1;
   for(var i = 0; i < self.bookmarks.length; i++) {
      if(self.bookmarks.at(i).get('address') === bkmrk.get('address')) {
         index = i;
         break;
      }
   }
   self.bookmarks.remove(bkmrk);
   self.removeFromView(index-1);
```

Server Side:

```
   module.exports = function(req, res) {
```

```
    if(!(req.user && req.body.group && req.body.address)) return;    //Sanitisation


    user.removeBookmark(req.user._doc.username, req.body.group,
                              {'address': req.body.address}, function(err){});
};
```

4.8 Page Parser

For several functionalities in Cloud Dial it is necessary to have access to the content being added. As such it is important to parse the page correctly. For this I use a tool called Diffbot. Diffbot figures out the structure and content of the web page without human intervention. It's an excellent tool, without which a considerable amount would have been lost developing an alternative. It is used in Cloud Dial to extract the body and title of the document, as well as all inline images.

Unfortunately Diffbot fails occasionally, so it was necessary to provide a fallback. In the rare case that it fails Node retrieves the content using the Request module and uses jsdom to extract the title and body.

```
exports.getPage = function(address, callback) {
  diffbot.article({'uri': address, 'summary': true}, function(err, response){
    if(!response.error) {     //If diffbot is successful
      callback(
        err,
        {
          title: response.title,
          body: response.text,
          summary: response.summary,
          media: response.media
        }
      );
    } else {     //Extract body and title
      try {
        request(address, function(err, response, body) {
          if(body) {
            jsdom.env({
              html: body,
                scripts: [
                  'http://code.jquery.com/jquery-1.5.min.js'
                ]
            },
            function (err, window) {
              var $ = window.jQuery;
```

```
                callback(err, { title: $('title').text(), body: $('body').text() });
              });
            } else {
              callback(err, body);
            }
          });
        } catch(err) {
          callback(err, undefined);
        }
      }
    });
  }
```

4.9 Tag Suggestion

Tag suggestion uses a combination of two approaches - popular tags and auto suggested tags.

4.9.1 Popular Tags

These are tags which have been previously chosen by users. Take for example the case where two users have added www.facebook.com. The first person uses the tags “social” and “friends”, so the second person will get these as suggestions. These tags are judged to be relevant by a human, so they are more likely to be relevant than automated suggestions. As such they will be suggested first.

4.9.2 Auto Suggested Tags

The Popular Tags approach works very well, however pages being added for the first time will not have these, and it's possible poor quality tags will be chosen by the first user. As such it is important to automatically suggest tags.

Cloud Dial uses Lucene to calculate the term weights of selected content. It calculates the weights based on frequency alone (ignoring stop words), with greater weighting given to terms that appear in the title. The terms with highest weights are suggested as tags. Though basic, this actually works very well.

```
for(int i = 0; i < TITLE_WEIGHT; i++) {
      body+=title;
}

Document doc = new Document();
doc.add(new Field("title", body, Field.Store.YES,
                          Field.Index.ANALYZED, TermVector.YES));
IndexWriter writer = new IndexWriter(index, config);
writer.addDocument(doc);
writer.close();

IndexReader reader = IndexReader.open(index);
```

```
    TermFreqVector [] vectors = reader.getTermFreqVectors(0);
    TopTermsList top = new TopTermsList();
    if(vectors != null) {
          TermFreqVector vect = vectors[0];
          String [] terms = vect.getTerms();
          int [] freqs = vect.getTermFrequencies();
          TopTermsList termlist = new TopTermsList();
          for(int i = 0; i < terms.length; i++) {
            TopTerm t = new TopTerm(terms[i], freqs[i]);
            termlist.add(t);
          }

          //Sort and take top
          Collections.sort(termlist);
          int numTop = amount > termlist.size() ? termlist.size() : amount;
          for(int i = 0; i < numTop; i++) {
                top.add(termlist.get(i));
          }
    }
```

Node.js performs quite poorly for computationally expensive tasks, so it made more sense to use Java for this feature. For the node.js <-> java communications this tool runs on a straight forward Java server. Node communicates with this server using the built in Net module. The page parsing functionality does its work, and sends it to the Java server. The server provides a thread from its thread pool for each connection. The term weights are calculated by this thread and sent to Cloud Dial.

4.9.3 Merging

The results of these two approaches are then merged. Any duplicates are removed.

```
function mergeWithoutDups() { //takes variable number of arguments
  var exists = {};
  var merged = new Array();
  for(var i = 0; i < arguments.length; i++) {     //For each array passed
    var currArray = arguments[i];
    for(var j = 0; j < currArray.length; j++) {
      var element = currArray[j];
      if(!exists[element]) {
        exists[element] = true;
        merged.push(element);
      }
    }
  }
  return merged;
}
```

4.10 Image Suggestion

Image suggestion uses a combination of three approaches. Popular Images, Inline Images and Auto Suggested Images. These are returned to the user in that order.

4.10.1 Popular Images

Image suggestion works in the same manner as tag suggestion. If an image has been used before it will be suggested to future adders. Again these will be ordered according to how many times they've been used previously. For this to work adequately it is assumed the same image is sourced from the same URL. Otherwise complex image analysis would be necessary.

4.10.2 Inline Images

As mentioned previously the page parsing tool returns all inline images. Research showed that these tend to be good representations of the content, so this was an obvious and straightforward addition.

4.10.3 Auto Suggested Images

Cloud Dial uses Google's image search API to provide additional suggestions. In order to effectively use this service a well formed query must be provided. Previously I queried the API using all tags suggested for the bookmark in question. This was returning some terrible results, as the chances of a relevant image being indexed under all those keywords was very low. I then queried it with various amounts of suggested tags, and with three I found the results to be much improved.

However the most effective approach proves to be a simply query with the pages title. Cloud Dial now uses this method whenever the pages title exists. In cases where it doesn't a query with the top three suggested tags is provided.

```
var terms = new Array();
if(page.title) {
  terms = page.title.split(' ');
} else {
  //Use three tags for the search, but if there's less use that amount.
  var numTags = tags ? (tags.length > 3 ? 3 : tags.length) : 0;
  for(var i = 0; i < numTags; i++){
    terms.push(tags[i]);
  }
}
google_images.getImageUrls(terms, callback);
```

4.10.4 Merging

The results for these three approaches are then merged in the same fashion as tags. It is assumed

that images with the same URL are duplicates.

4.11 Description Suggestion

To provide a description suggestion Cloud Dial summarises the content being added. Classifier 4J's SimpleSummarizer is used to provide these summaries.

SimpleSummariser is actually quite poor, so improvements were necessary. It turned out the algorithm being used was extremely basic. It simply returned any sentences containing all terms from the title. Using some of the approaches detailed in the research section it was possible to extend it's capabilities. Most notably I used Luhn's Keyword Cluster Method, title term weighting and location bias, which made considerable improvements.

Like the tag suggestions functionality, summarisation is quite computationally expensive. As such it is developed using Java. It works in the same thread as the tag suggestor, and returns the summary along with the tags. These are transferred using JSON.

```
JSONObject titleAndBody = new JSONObject(doc);
String title = titleAndBody.getString("title");
String body = titleAndBody.getString("body");

Summariser summariser = new Summariser();
String summary = summariser.summarise(body, title, 1);

TagSuggestor tagSugg = new TagSuggestor("stopwords.txt");
TopTermsList tags = tagSugg.suggest(body, title, 10);

JSONStringer stringer = new JSONStringer();
stringer.object();
stringer.key("summary");
stringer.value(summary);
stringer.endObject();
String jsonSummary = stringer.toString();
jsonSummary = jsonSummary.substring(1, jsonSummary.length()-1);

stringer = new JSONStringer();
stringer.object();
stringer.key("tags");
stringer.array();
for(TopTerm tag : tags) {
	stringer.object();
	stringer.key("tag");
	stringer.value(tag.getTerm());
	stringer.key("amount");
	stringer.value(tag.getFreq());
	stringer.endObject();
}
stringer.endArray();
stringer.endObject();
String jsonTags = stringer.toString();
jsonTags = jsonTags.substring(1, jsonTags.length()-1);

String output = "{" + jsonSummary + "," + jsonTags + "}";
```

4.12 Group Suggestion

In order to aid bookmark addition Cloud Dial tries to work out what group each bookmark should be added to. The approach used for this is straightforward and effective. When a user selects a bookmark for addition Cloud Dial suggests tags for that bookmark. Cloud Dial checks how many times these tags have occurred in each group, and selects the group with the highest score. If the scores are equal it simply selects the default group.

Example

Group1, Social:

Bookmark - 'www.facebook.com'. Tags - 'social', 'friends'

Bookmark - 'www.myspace.com'. Tags - 'social', 'music'

Group2, Technology:

Bookmark - 'www.techcrunch.com'. Tags - 'tech', 'news'

Bookmark being added:

'www.bebo.com'. Suggested tags – 'social', 'friends'

Scores:

Group1 – 3

Group2 - Score: 0

Suggestion:

Group1, Social

```
function suggestGroup(username, suggTags, callback) {
  var groupScores = new Array();
  user.getTagAmountsByGroup(username, function(groupTags) {
    user.getGroupNames(username, function(err, groupNames){
      for(var i = 0; i < groupNames.length; i++) {
        var score = 0;
        suggTags.forEach(function(tg) {
          if(groupTags[groupNames[i]][tg]) {
            score+=groupTags[groupNames[i]][tg];
          }
        });
        groupScores.push(score);
      }
```

```
      var suggGroup = groupNames[utils.getMaxIndex(groupScores)];
      callback(err, suggGroup);
    });
  });
}
```

4.13 Bookmark Searching

Users search their bookmark collection based on previously selected tags. It is a simple boolean search - searching “tag1" will return all bookmarks in their collection with tag “tag1". Searching “tag1, tag2" will return all bookmarks with both “tag1” and “tag2”.

```
grp.bookmarks.forEach(function(bkmrk){
  var match = false;
  for(var i = 0; i < tags.length; i++) {
    match = false;
    for(var j = 0; j < bkmrk.tags.length; j++) {
      if(tags[i] === bkmrk.tags[j]) {
        match = true;
        break;   //We found a match, check the next tag
      }
    }
    //If any tag searched for doesn't match we stop checking the current bookmark
    if(!match) break;
  }

  if(match) {
    bookmarks.push(bkmrk);
  }
});
```

When a user makes a search Cloud Dial also suggests tags to refine the search. The other tags from all bookmarks in the search space are suggested. These are suggested in order of popularity.

```
//Returns all tags from collection in order of popularity
var tempTags = {};
this.bookmarks.forEach(function(bkmrk){
  var tags = bkmrk.get('tags');
  if(tags) {
    tags.forEach(function(tg){
      if(!tempTags[tg]) {           //If tag is not added
        tempTags[tg] = { 'tag': tg, 'amount': 1 };
```

```
      } else {
        tempTags[tg].amount++;
      }
    });
  }
});

//Creating final array
var tgs = new Array();
for(var prop in tempTags) {
  tgs.push(tempTags[prop]);
}
tgs.sort(function(a,b){ return a.amount > b.amount ? -1 : 1 });
return tgs;
```

4.14 Privacy

Bookmarks and groups are simply public or private. If private they are only viewable when a user logs in. Otherwise a users' public groups and bookmarks are viewable at /user/<username>. If a group is private all bookmarks contained within will not be viewable even if they are public.

```
getPublicBookmarks: function(username, callback) {
  this.getBookmarks(
    username,
    function(grp) {     //Bookmark filter callback
      var bkmrks = new Array();
      grp.bookmarks.forEach(function(bkmrk){
        if(!bkmrk.private) {
          bkmrks.push(bkmrk);
        }
      });
      grp.bookmarks = bkmrks;
      return !grp.private;
    },
    callback            //Callback for complete groups
  );
}
```

It is hoped users will make this page their browser's homepage on all devices. Doing so would undoubtedly increase productivity, by facilitating quick access to bookmarked sites.

4.15 Facebook Sharing

When a user adds a publicly viewable bookmark it is posted on their Facebook account. This will enable users to share their online activities with their friends, or not at all if they so choose. It is developed using Facebook's Open Graph API.

Open Graph applications are generally concerned with specific object interaction, eg. “Michael Smith has created a recipe on CookMe”, and Open Graph is developed with this type of interaction in mind. However with Cloud Dial a user can potentially interact with any type of object. If a user adds an page on BBC its object type is “article”, if they add a page on Youtube its object type is “video”. It is necessary to know the object type in question before posting it to Facebook, and only previously specified objects in the application settings can be interacted with. Cloud Dial currently supports all built in object types (website, blog, article, video etc.)

```
FB.api(
  '/me/clouddial:add?website=' + bookmark_model.get('address'),
  'post',
  function(response) {}
);
```

4.16 REST API

Cloud Dial tries to strictly conform to the REST constraints. The correct REST verbs are used for all client server communications, the correct status codes are sent in responses from the server, and the URI contains only the nouns relevant to the action in question.

Web framework Express.js is used to provide resourceful routing. It made it easy to separate the model, view and routing code. Detailed below is Cloud Dial's REST API -

Bookmarks

`GET /user/user_id`

Returns a users groups and bookmarks.

Example Response

If the user exists

```
{
  groups: [
    {
      name: 'home',
      bookmarks: [
        {
          address: 'www.facebook.com',
          title: 'Facebook',
          tags: ['social', 'friends'],
```

```
            description: 'Social networking website',
            private: false
          }
        ],
        default: 'true'
      }
    ]
  }
```

If the user doesn't exist

```
Status code: 404
```

Search

`GET /user/user_id/bookmarks?`

Returns all user's bookmarks that match the provided tags.

Arguments

`&tags`	Tags to search (comma delimited)

Example Response

```
bookmarks: [
  {
    address: 'www.facebook.com',
    title: 'Facebook',
    tags: ['social', 'friends'],
    description: 'Social networking website',
    private: false
  }
]
```

Adding Bookmark

`PUT /bookmark?`

Adds a bookmark with the provided arguments to a selected group.

Arguments

`&title`	Title of page being added (optional)
`&address`	URL of page being added (required)
`&imgAddress`	URL of representative image (optional)
`&tags`	Comma delimited tags (optional)

`&description`	Bookmark description (optional)
`&private`	Privacy setting (public if not provided)
`&group`	Containing group name (required)

Example Response

If successful

```
Status Code: 200
```

If not successful

```
Status Code: 400
```

Deleting Bookmark

`DELETE /bookmark?`

Deletes bookmark with provided URL for selected group.

Arguments

`&address`	URL of page being deleted (required)
`&group`	Containing group name (required)

Example Response

If successful

```
Status Code: 200
```

If not successful

```
Status Code: 400
```

Adding Group

`POST /user/`*`user_id`*`/group?`

Deletes bookmark with provided URL for selected group.

Arguments

`&name`	Name of group being created (required)
`&private`	Privacy setting (public if not provided)

Example Response

If successful

```
Status Code: 200
```

If not successful

```
Status Code: 400
```

5 Problems and Resolution

5.1 Callback Pyramids

Due to it's asynchronous nature callbacks are widely used in JavaScript, however there are many recognised issues with them. All to often they lead to deeply nested callback structures, and Cloud Dial's code base was beginning to fall prey to this.

```
user.getBookmarkImages(req.query.address, function(suggImgs){
  user.getBookmarkTags(req.query.address, function(suggTags){
    google_images.getImageUrls(utils.arrayToString(suggTags), function(imgUrls){
      top_terms_client.getTopTerms(req.query.address, function(topTerms){
        res.render(
          'savescreen',
          {
            title: req.query.title,
            address: req.query.address,
            pop_images: suggImgs,
            goog_images: imgUrls,
            pop_tags: suggTags,
            top_tags: topTerms
          }
        );
      });
    });
  });
});
```

There are several popular remedies used to solve this problem, namely futures and promises. However using either would have resulted in a significant amount of code re-factoring, so I needed to find a better solution. This came in the form of Async by Caolan McMahon. This improved the above code considerably.

```
async.parallel(
  {
    groupNames: function(callback){
      user.getGroupNames(req.user._doc.username, callback);
    },
    popularImages: function(callback){
      bookmark.getBookmarkImages(req.query.address, callback);
    },
    inlineImages: function(callback){
```

```
    var media = page.media, inlineImages = new Array();
    if(media) {
      for(var i = 0; i < media.length; i++) {
        if(media[i].type === 'image') {
          inlineImages.push(media[i].link);
        }
      }
    }
    callback(null, inlineImages);
  },
  suggestedImages: function(callback){
    var terms = new Array();
    if(page.title) {
      terms = page.title.split(' ');
    } else {
      var numTags = tags.length > 0 ? (tags.length > 3 ? 3 : tags.length) : 0;
      for(var i = 0; i < numTags; i++){
        terms.push(tags[i]);
      }
    }
    google_images.getImageUrls(terms, callback);
  },
  suggestedGroup: function(callback){
    if(!req.query.group) {
      suggestGroup(req.user._doc.username, tags, callback);
    } else {
      callback(null, req.query.group);
    }
  },
  description: function(callback){
    bookmark.getBookmarkDescription(req.query.address, callback);
  }
},
function(err, results)
{
  res.render(
    'savescreen',
```

```
      {
        'title': page.title,
        'address': req.query.address,
        'groups': results.groupNames,
        'images': mergeWithoutDups(results.popularImages, results.inlineImages,
                                   results.suggestedImages).splice(0,10),
        'tags': tags.splice(0,10),
        'sugg_group': results.suggestedGroup,
        'description': (page.summary ? page.summary : results.description)
      }
    );
  }
);
```

Not only is it a lot easier to read, but it's also performs better. The previous code ran sequentially. Using async.js it now takes full advantage of node.js's asynchronous nature.

5.2 Open Graph Object Types

As mentioned in the implementation section Open Graph applications are concerned with specific object types. As such it is assumed that the poster knows what object type to specify when calling the API. This proved to be a problem for Cloud Dial because a user can potentially interact with any object type. The following hack by passes this.

```
FB.api(
  //Assume it's a website
  '/me/clouddial:add?website=' + bookmark_model.get('address'),
  'post',
  function(response) {
    if(response.error && response.error.code == 3502) {
      var objType = response.error.message.split('\'')[1];
      //message format 23/5:
      //(#3502) Object at URL http://example.com/ has og:type of 'article'
      //0:[(#3502) Object at URL http://example.com/ has og:type of ] 1:[article]
      FB.api(
        '/me/clouddial:add?' + objType + '=' + bookmark_model.get('address'),
        'post',
        function(resp){
          finished();
        }
      );
```

```
      } else {
        finished();
      }
    }
  );
```

5.3 Realtime bookmark addition

The development itself was quite time consuming. On top of the initial learning curve, I also spent quite a while resolving the problem of different session ids. As expected yhis value is different each time a user logs in, even if they are logged in to the same account on different machines. Unfortunately socket.io made it difficult to access the session variable, however using an extended module, socket.io-sessions; I managed to get access it. This made it possible to group all session ids by username. Now whenever a user adds a bookmark all session ids stored under their username are sent the bookmark information.

6 Testing

6.1 Automated Testing

A number of automated tests were developed to back the major components of Cloud Dial. All JavaScript tests were developed using Mocha on the client and server side, while all Java tests were developed using JUnit.

Users

Pre-condition: User 'test' exists.

Post-condition: User 'test' no longer exists.

Test no.	Steps	Data	Expected Results
1	- Retrieve user 'test'	Username → 'test'	User 'test' is found.
2	- Retrieve user 'fake'	Username → 'fake'	User 'fake' is not found.
3	- Add bookmark to test's account. - Retrieve bookmark.	Group → 'home' Address → 'www.test.com' Image Address → 'www.testimage.com' Tags → 'Test1', 'Test2' Private → False	Bookmark with all it's information is added to group 'home' of user 'test'.
4	- Add public bookmark to test's account. - Add private bookmark to tests account. - Retrieve test's public bookmarks	Group → 'home' Address → 'www.test.com' Image Address → 'www.testimage.com' Tags → 'Test1', 'Test2' Private → False Group → 'home' Address → 'www.test2.com' Image Address → 'www.testimage.com' Tags → 'Test1', 'Test2' Private → True	Only test's public bookmarks are retrieved.
5	- Add extra group 'another' to test's account. - Add bookmark to group 'home'. - Add bookmark to group 'another'. - Retrieve all of test's bookmarks.	Name → 'another' Private → false Group → 'home' Address → 'www.test.com' Image Address → 'www.testimage.com' Tags → 'Test1', 'Test2' Private → False	Group 'another' has been added, and contains bookmark www.test2.com. Group 'home' still exists and contains bookmark www.test1.com.

		Group → 'another' Address → 'www.test2.com' Image Address → 'www.testimage.com' Tags → 'Test1', 'Test2' Private → False	
6	- Add group 'another' - Retrieve group names.	Name → 'another' Private → false	Both 'home' and 'another' are returned.
7	- Retrieve bookmarks of non-existent user	Username → 'fake'	No bookmarks are returned.
8	- Remove bookmark from users collection - Retrieve users bookmarks	Username → 'test' Group → 'home' Address → 'www.test.com'	www.test.com has been removed from test's collection.
9	- Add bookmarks to different groups with a variety of tags. - Retrieve the group breakdown of tags.	Group → 'home' Address → 'www.test.com' Tags → 'test1', 'test2' Group → 'home' Address → 'www.test2.com' Tags → 'test1', 'test3' Group → 'home' Address → 'www.test.com' Tags → 'test1', 'test3' Group → 'home' Address → 'www.test.com' Tags → 'test1', 'test3'	The tag amounts are correct for each tag in each group.

Bookmarks

Pre-condition: Bookmark with these attributes exists – address: 'www.test.com', image address: 'www.testimage.com', suggested tags: 'test1', 'test2', description: 'this is a test bookmark'.

Post-condition: Bookmark 'www.test.com' is removed.

Test no.	Steps	Data	Expected Results
1	- Retrieve bookmark www.test.com	Address → 'www.test.com'	Bookmark is found with all information saved.
2	- Retreve bookmark www.fake.com	Address → 'www.fake.com'	Bookmark is not found.
3	- Add bookmark www.test.com with different representative image.	Address → 'www.test.com' Image Address → 'www.testimage2.com'	Bookmark is found and has two representative images, each with an amount of 1.

	- Retrieve bookmark www.test.com		
4	- Add bookmark www.test.com with the same representative image. - Retrieve bookmark www.test.com	Address → 'www.test.com' Image Address → 'www.testimage.com'	Bookmark is found and has one representative image. The amount value of this image is 2.
5	- Add bookmark www.test.com with user defined tag. - Retrieve bookmark www.test.com	Address → 'www.test.com' Tags → 'test3'	Bookmark is found and has one popular tag – test3. The amount for this tag is 1.
6	- Add bookmark www.test.com with user defined tag. - Add bookmark www.test.com with the same user defined tag. - Retrieve bookmark www.test.com	Address → 'www.test.com' Tags → 'test3' Address → 'www.test.com' Tags → 'test3'	Bookmark is found and has one popular tag – test3. The amount for this tag is 2.
7	- Add bookmark www.test.com with representative image. - Add bookmark www.test.com with the same representative image. - Retrieve the images of www.test.com	Address → 'www.test.com' Image Address → 'www.testimage2.com' Address → 'www.test.com' Image Address → 'www.testimage2.com'	Images are returned in order of popularity. The first is www.testimage2.com. The second is www.testimage1.com.
8	- Add bookmark www.test.com with tags. - Add bookmark www.test.com with tags. Ensure one is the same as the previous bookmark's. - Retrieve the tags of www.test.com	Address → 'www.test.com' Tags → 'test1', 'test2' Address → 'www.test.com' Tags → 'test2', 'test3'	Tags are returned in order of popularity. The first is 'test2'.

Google Images

Test no.	Steps	Data	Expected Results
1	- Search with genuine query.	Query → 'will smith'	8 images are returned.
2	- Search with empty query.	Query → ''	No images are returned.

Page Parser

Test no.	Steps	Data	Expected Results
1	- Parse genuine web page.	Address → 'www.test.com'	Body of www.test.com is returned.
2	- Attempt to parse incorrect input.	Address → 'asdfsdfsdf'	Undefined is returned.
3	- Parse non existing web page.	Address → 'www.idefinitelydontexist.com'	Undefined is returned.

Tag Suggestor

Test no.	Steps	Data	Expected Results
1	- Tag real content.	Body → 'Cloud Dial is a web application that allows users to store, manage, retrieve and share their bookmarks anywhere, any time, straight from their browser...' Title → " Number of tags → 10	Tags are returned in numerical order – 'will, bookmarks, bookmark, cloud, dial, browser, users, web, browsing, devices'
2	- Test content with less than 10 possible tags	Body → 'Cloud Dial allows users to store bookmarks, manage them, and retrieve and share these bookmarks anywhere.' Title → " Number of tags → 10	Less than 10 tags are returned.
3	- Tag empty content	Body → " Title → " Number of tags → 10	No tags are returned.

Summariser

Test no.	Steps	Data	Expected Results
1	- Summarise proper content.	Body → 'Cloud Dial is a web application that allows users to store, manage, retrieve and share their bookmarks anywhere, any time, straight from their browser...'	Summariser completes and outputs sentence - “Cloud Dial is a web application that allows users to store, manage, retrieve and share their bookmarks anywhere, any time, straight from their

		Title → " Number of sentences → 1	browser”
2	- Summarise no content	Body → " Title → " Number of sentence → 1	No summary is outputted.

6.2 Usability Testing

Cloud Dial's interactivity design is user-centred. As such it was important to evaluate its effectiveness by testing it on potential users. Usability tests where carried out with ten people of varying technical abilities. Prior to attempting the tasks they were informed of the purpose of Cloud Dial, and the motivations for it's development.

Here are the results of the initial tests. The design will be modified based on these, and the testing process repeated.

6.2.1 Test Results

Satisfaction is scored out of 5.

1. You have arrived at the Cloud Dial homepage, and you want to sign up. Add the Cloud Dial bookmarklet and sign up through Facebook.

Average errors: 1

Average satisfaction rating: 3.75

Comments:

- Bookmarklet should be bigger.
- 'Bookmark bar' instead of 'toolbar'.*
- Instructions on image.
- Clearer explanation of bookmarklet's capabilities.*

2. You see an empty account page with one group, Home. Select a website to add, and launch the bookmark addition screen.

Average errors: 1.125

Average satisfaction rating: 3.875

Comments:

- Make the add symbol bigger/clearer.*
- Cursor should start in addition box.
- Clear afterwards.

- Don't require 'http://www.' at the start of links.

3. The bookmark addition screen has appeared with the name of the website you selected, as well as several other options. take a moment to check all of these. Now select an image that will represent your bookmark. If you are not happy with the suggested images choose your own.

Average errors: 1.125

Average satisfaction rating: 3.25

Comments:

- Fix changing manually added image on hover.
- Title over image URL box.
- Scale image correctly
- Provide access to Google image search for manual selection

4. You have selected an image, and see the next option as you move down the page is tagging. Choose some tags for the bookmark (these may be your own and/or some of those suggested).

Average errors: 0.125

Average satisfaction rating: 4.25

Comments:

- Suggestions above input box.*
- Limit tag length (long tags look horrible)
- Handle comma separated better.

5. You are happy with all of the tags you have chosen for your bookmark. You notice the next option, description, is already populated. If you are happy with this description move on, otherwise you may change it.

Average errors: 0

Average satisfaction rating: 3.75

Comments:

- Make it clear that it's optional

6. You have reached the bottom of the addition screen. Select a privacy option and group, and finally save it.

Average errors: 0

Average satisfaction rating: 4.125

Comments:

- Explain why you should make a bookmark private.*
- Indication that bookmark is saving before closing.

7. Now that you are finished check that your bookmark has being added on your account page. take a moment to check all the information added with your bookmark.

Average errors: N/A

Average satisfaction rating: 3.75

Comments:

- Change 'Selection Tags' to '<group name> Tags'/'Search tags'.*
- Centre image.
- Have 'All' group.*
- Add sorting options

8. Before adding some more bookmarks add groups to your Cloud Dial account page. The privacy options are your choice.

Average errors: 0.375

Average satisfaction rating: 3.675

Comments:

- Cursor in box on launch.
- Tool tip on add button.
- Clear previously added group name.
- Return to submit.
- Explain what exactly privacy means.
- Add group button could be clearer. (Possible colour differentiation)
- All group reorganisation.

9. Repeat steps 2 to 8 several times.

Average errors: N/A

Average satisfaction rating: 3.25

Comments:

- Shorten long tags

- Use scrollbar for tag selection.

10. You have added several bookmarks now. Have the all appeared on your Facebook account?

Average errors: N/A

Average satisfaction rating: 4.25

Comments:

- Do not post private bookmarks.

11. You now have several groups and bookmarks on your Cloud Dial account page. Try searching for some of them.

Average errors: 0.5

Average satisfaction rating: 3.875

Comments:

- Predict tag

12. Trying deleting some of your bookmarks. Can you verify they are in fact deleted?

Average errors: 0.5

Average satisfaction rating: 4

Comments:

- Confirm to delete.*
- Clearer 'x' icon.

13. You saved descriptions with your bookmarks. Can you find them?

Average errors: 0

Average satisfaction rating: 4.75

Comments:

- Not as close to 'x'.
- 'i'/Info icon instead of question mark.
- More visible.*

Average time: 35 minutes

Average rating: 6.3

% that would use Cloud Dial again: 100%

Average age: 27

Average 'Technology Savviness' rating: 7.5.

Overall comments:

- Add tool tips*
- More colourful
- Add ability to edit

All comments with '*' were mentioned by several testers.

6.2.2 Conclusions

Overall users seem somewhat content with the interactivity design of Cloud Dial, however several issues undoubtedly need attention - bookmark tags and image selection in particular. On a more positive note; the reaction to the application and ideas behind Cloud Dial received full praise. All testers said they would use the application again (even if they rated it badly!), and have offered to test it again if needed.

7 Future Work

There is massive scope for expansion of Cloud Dial. The architecture in place is stable, and scalable, ensuring it is prepared for further work.

Here are some possibilities. I'm sure you can think of more -

7.1 Bookmark Editing

This is an obvious addition. Users should be able to edit created groups and bookmarks.

7.2 Mobile Interface

The current interface works well on desktop PCs and tablets. However to ensure Cloud Dial is truly accessible everywhere a mobile interface is necessary. This is a reasonably trivial development, involving creation of mobile specific stylesheets, and some business logic changes based on a devices user agent string.

7.3 Subgroups

This was mentioned in the functional specification. Currently users can create groups, however it would be an excellent addition to enable creation of groups within groups.

7.4 Bookmark Suggestions

Like sub-group creation this was mentioned in the functional specification. This would be a nice addition to Cloud Dial, but it would take a little code refactoring. Suggestions would likely be made based on a users current bookmarks. Those in Cloud Dial's bookmarks collection that are most similar to these would be suggested. However in order for this to work well a tags collection is likely needed. It is undoubtedly worth the effort though.

7.5 Customisation

Users like to have control over their content. Some customisations that would work well are drag and drop reordering, background selection and dial customising. The ability to use custom style sheets should also be available.

7.6 Bookmark and Tag Ordering

Users should have the option of ordering their bookmarks alphabetically and chronologically. This should be possible at a group level, and for search results.

Tags should be capable of being order alphabetically and by amount.

7.7 Browser Add-ons

Bookmarklets provide a cross browser solution for convenient bookmark addition, however they confuse a lot of users, most of whom are not aware of their existence. Having browser add ons and extensions should ease this confusion.

7.8 Mobile Apps

For similar reasons as add-ons and extensions, mobile apps should be available for those not altogether familiar with mobile web applications.