National University of Singapore
School of Computing
CS3216: Software Development on Evolving Platforms
AY2011/2012, Semester 1

Assignment 2: **Mobile Cloud Application**

Issue date: 29 August 2011
Due date: 24 September 2011

General Overview

In the recent years, mobile cloud computing has been one of the hottest topics regarding technological advancement. So what exactly is mobile cloud computing and why is it given so much attention?

Smartphones and tablets are evidently becoming the norm these days. Thanks to these technologies, people have found new ways to spend their time more fruitfully and it is only natural that they keep trying to do more with their mobile devices. Mobile cloud computing is a crossover between mobile web and cloud, offering greater possibilities in terms of power and productivity.

Mobile web refers to browser-based web applications for mobile devices. These applications, compared to native ones, used to be handicapped by the lack of features such as offline storage and support for hardware such as cameras. To address the situation, the W3C has proposed a list of standards that browsers can follow to give developers a consistant API to access the features that they were not able to before, across many different devices. Some of these standards have already been implemented while others are on their way, but in the near future we can expect the gap between native and mobile web applications to narrow.

Cloud computing is where data and processing power are accessed from a pool of shared online resources known as the cloud. You can visualise it as a super computer containing all the applications and data belonging to their users. A user can run his applications and process the data in the cloud using any connected device as a remote controller.

Combining their characteristics results in mobile cloud applications with the following strengths:

1. **Availability**. With all data and configurations stored in the cloud, the user will be able to enjoy the same experience any time and anywhere using any mobile device.
2. **Productivity**. What we previously could do only at home or in the office can now be done on the go. We are now able to read our emails the very moment one is received, edit documents or even catch up with friends over Facebook during a bus trip.
3. **Device independent**. Want to replace your iPhone with the latest Android? A web application runs in any modern browsers so there is no need to worry if your favourite application has yet to be ported to other platforms.
4. **Low hardware requirements**. All heavy computation is offloaded from the client to the servers, reducing its role to an interface for sending commands to the cloud. In addition, parts of the data backed up in the cloud can be served on demand. The mobile device therefore, does not require very strong processing power and large storage capacity.

Grading and Admin

This assignment can be done in groups of three or four students. If you are unable find yourself a group, you will be randomly assigned to one.

We will not be providing step-by-step instructions. Instead, we will only list the key concepts you need to master (also known as ”milestones”). We will also provide some related tips, references and a little bit of help to get you started. These milestones constitute 70% of the assignment's grade.

Like the previous assignment, if you find that some of the proposed milestones do not make sense for the application you intend to build, you can petition to replace them with some other deliverables. You are to explain why we should agree to your petition, and submit your petition at least one week before the assignment is due. Your petition is subjected to approval.

While the milestones may be easy to meet, simply meeting them will not give you the maximum credit. We ask for quality submissions, not run-of-the-mill work.

To score the coveted remaining 30%, we would like to see how you can put your creativity to the test and explore all pores of your originality tissues. We choose not to restrict your potential by insisting on any particular sequence. We strongly recommend that you blow us away with your creativity.

Please do not hesitate to approach the friendly CS3216 staff if you need further assistance. If you have questions, please post them in the IVLE Discussion Forum.

Objectives

In this assignment you are required to deploy a mobile cloud application. Your goal is to demonstrate that you can design your own RESTful API, followed by implementing a HTML5 client and PHP REST server that communicate using your API. But as before, you are free to decide what your application does as long as nothing illegal or immoral is involved.

Extra credit will be awarded to groups with creativity, but do note that feature additions should “make sense” and bring value to your application.For example, geolocation services could be the latest and coolest feature in HTML5 but it may not contribute to the user experience in a document editor. In fact, implementing features for the sake of doing so may even work against you by confusing the user.

Remember, your goal is not to do a lot of work. Your goal is to use this opportunity to “make a difference”. If you can make a difference by just doing a little bit of work, that's fine with us.

Before you begin, do spend some time understanding the requirements for the assignment. Also, try to get some sleep before you start on this assignment. To help you avoid losing sleep, we have included a detailed grading scheme at the end of this handout.

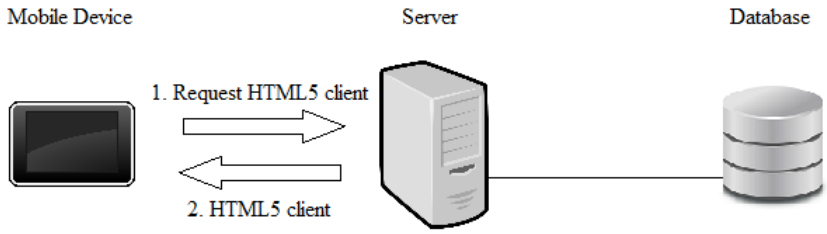
Introduction

Welcome! This assignment comprises of 3 weeks of intensive learning that provides you with another opportunity to express your creativity. Without further delay, we shall look at what makes a mobile cloud application.

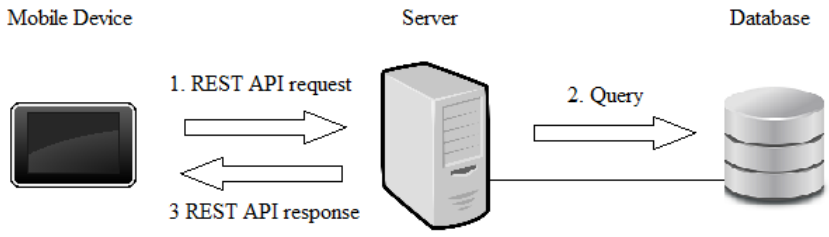
A mobile cloud application is typically a combination of the following components:

1. **Clients**. Clients reside on mobile devices belonging to users, including netbooks, smartphones as well as tablets recently made popular by Apple. It could be (1) native applications, which run only on their intended platforms or (2) HTML5 web applications with the potential to work on any device with a HTML5-enabled browser. The client application is generally light-weight and unintelligent, doing nothing more than its sole purpose of providing the user with an interface to view and manipulate data.
2. **Server**. The server, where most actual and intensive computation is done, sits and waits for jobs from clients to come in. After processing these requests, the server may then inform the client of its success or return relevant information to be shown to the user.
3. **Database**. The database does exactly what we would expect of it in any other context, which is to store all related data of the application.

Depending on the size of the service, different amounts of hardware may be deployed to keep up with the demand of its users. Obviously, large services will require more than a single server and database. There are many ways to configure multiple servers and databases to work together so that the service scales. Several server instances can be run simultaneously while using load balancers to split the work evenly among them. Data can also be replicated over several databases and/or distributed based on pre-defined hash functions for higher reliability and availability. Such are issues to consider as you extend your reach to more users but in this assignment we shall start small. In its simplest form, a mobile cloud application can be hosted with the server and database sharing an AWS instance.



Normally, users would download the clients from the online application stores for their respective platforms. In our case, the workflow differs a little from native applications. Since our client is created with **HTML5** and supporting web technologies such as **JavaScript** and **CSS3**, it is essentially a normal web page with a few quirks. During the user's first visit to the site, we will tell the browser to quietly download and save the program for future use. The **Application Cache** is how we will instruct it to retain all resources required for this "web page" to run in the absence of an internet connection. When used in conjunction with **Web Storage** or **Web SQL Database**, which allow storage of data on the local file system, your application can function like a native one.



After which, the client can operate with limited functionality when it is offline and communicate with the server using **AJAX** calls while it has access to the internet. Requests typically take the form of **JSON** or **XML** formatted messages and they contain details of a job to be processed by the server, such as querying the database for some information or to update its records. The server then replies with a similarly formatted message response, which the client is responsible for decoding and notifying the user of. **PHP** has inbuilt functions to support database queries as well as JSON messages.

Fret not if you are not already familiar with the different technologies indicated in bold font. More about them will be discussed in later sections.

Reminder:	Please read the entire assignment before thinking about what you want to develop. It may give you a clearer idea of how all the parts come together and also a better understanding of the strengths and weaknesses of a mobile cloud application.

Phase 1: Design

Idea Generation

> No charge for awesomeness!
>
> – Kungfu Panda

As an aspiring entrepreneur of CS3216, you have a mind full of big ideas eagerly waiting to see the light of day. After a long discussion with the team about the unlimited potential of your new product, you are now more sure than ever that this is going to be the next big thing. You expect more users than Facebook and Twitter combined and to keep this many people waiting any longer is simply evil. . .

But before you jump straight into planning the specifications of your awesome application, a very important question to ask yourself is, “Does my application solve any problems for the users?” An application that has many superfluous features does not make it any useful. Just because your application has a cool concept or uses the latest technology does not guarantee that initial users who joined out of curiousity will stay on. On the contrary, if a less fancy but more down-to-earth solution makes the user’s life a lot easier, it is more likely that the user will be retained. Solve a problem that people care about, solve it well, and fans of your application will naturally accumulate.

Milestone 0:	Describe the problem that your application solves.

As exciting as developing on a new platform may seem, it also has problems associated with being new. Why would you want to develop on the tablet platform instead of another more established platforms, such as Facebook?

Building a killer application requires more than just technical skills. In CS3216, we expect you to think very hard about what you’re trying to do. You should not be building an application just because you need to submit this piece of homework.

You should choose an application that truly exploits the potential of the platform. For example, you should have a good answer to the question of why developing on other platforms is not the best solution to fulfill your application's objectives.

In CS3216 (and life in general), execution matters. Identifying a good idea is the first step, deciding on the path of development to take, that ensures maximum success for your application, is the next. Choosing the most suitable platform for your application is also crucial in your execution. Thus, we expect you to come up with a good reason for it.

Milestone 1:	Describe your application and explain how you intend to exploit the characteristics of mobile cloud computing to achieve your application's objectives, i.e. why does it make most sense to implement your application as a mobile cloud application?

Unlike the Facebook application you have developed in the first assignment, your mobile cloud application does not have a ready-made social network to leverage on. It is no good to have a killer app that nobody uses. Hence, you will also be expected to think a little harder about how you plan to "market" your app to potential users. You must identify your target users, determine the relevance of your application to them (i.e. why should they care about your application) and explain how you plan to reach out and persuade them to use your application.

In order to promote the use of your application, good marketing strategies are crucial in raising awareness of your application among the targeted users. After introducing potential users to your application, how would you try to persuade them to continue using the application, and perhaps, even share or introduce it to others? What value do the users derive from using your application?

Ideally, you should also think of ways to provide motivation for users to share your application with their friends. The application should be designed such that while individual users may derive some value using your application, it is in their interest to promote your application (e.g. A reward scheme can be set up to give out benefits for referrals).

The number of users with mobile devices is evidently growing rapidly in the recent years. Given this trend, how would you take advantage of this growth to increase the number of people using your application?

Milestone 2:	Describe your target users. Explain how you plan to promote your application to attract your target users.

Now that you have a solid idea of what you want to build, it is a good time to pick a suitable name for your mobile cloud application if you haven't already done so.

Milestone 3:	Pick a name for your mobile cloud application. *(Not graded)*.

Database Planning

Make sure you take some time to plan a good schema design. Having to change the schema and code later can be a painful process, more so if you already have a considerably large user base. Refer to the previous assignment if you need a quick recap on relational databases. Remember the principle of **design once use forever**.

> **Milestone 4:** Draw the database schema of your application.

RESTful API

Much as we would love to immediately begin creating the application, there is still one final thing we need to draw up plans for, the RESTful API itself. Being the bridge between the client and server, both sides should be very clear of the specifications to avoid unnecessary confusion during development. If you only decide on parts of the interface while you program the application, you risk having an inconsistent API. It pays off to plan this well, who knows, your application may become so prominent that you want to open the API to third-party developers.

Representational State Transfer, REST in short, is an architecture for networked applications. Being one of the simplest architectures to deploy, it is a popular choice for many web services. REST is a client-server architecture where a client initiates a request to the server to be processed, and receives a response with updated data. It is stateless, meaning that the server does not maintain the state of the client and every message to the server must contain all necessary information to process the request. Not having persistent connections or complex states means that the service can be setup using simple web servers, and requests can be distributed across multiple servers for greater scalability. You may have already noticed that the Facebook Graph API falls under this category as well.

Requests and responses can take many forms. `XML`, `JSON`, `multipart/form-data` and `application/x-www-form-urlencoded` are just a few common examples that come to mind. Note that the request and response may not necessarily be in the same format. For example, when uploading a photo using the Graph API, `multipart/form-data` is sent while the reply containing the photo id is in JSON format. We now take a quick detour to explore the JSON format since it is one of the easier ones to handle.

JavaScript Object Notation or JSON is text-based data-interchange format. Consider your current scenario where you are working with an object in your client using JavaScript and you need to send it over the network to be processed by the server. JSON is a set of rules to encode and decode data to and from a universally recognized string format so it can be used with any programming language which implements these rules. JSON works on any object made up of key/value pairs and/or ordered lists of values. The resultant string is also human-readable unless encoded binary files are involved. The following JSON string could be a response from a social network's REST API to a request for information on the user with id = 0, where { } represents an object and [ ] an array.

```
{
  ”id”: ”0”,
  ”name”: ”James”,
  ”tel”: ”61234567”,
  ”mobile”: ”87654321”,
  ”friends”:
```

```
  [{
    "id":"1",
    "name":"Andy"
  },
  {
    "id":"2",
    "name":"John"
  }]
}
```

JSON is already supported by all modern browsers and server-side scripting languages so you do not need to implement it yourself, but if you want to know more details, visit `http://www.json.org/`.

Coming back to the API, one way to start planning it is to casually write down a list of functionalities (which require server interaction) that you would like the client to have (e.g send a private message, buy a product, leave a comment). Try to group the related ones together (e.g add / remove a friend).

Next, for every operation, you establish a triplet which describes it:

1. **Request method + relative URL**:

   When an address is entered into the navigation bar, the browser performs a GET request to retrieve the page located at the URL. However, a POST request is made instead when we submit a registration form on the same page. If you haven't realised, the HTTP request method describes the action we want to perform and the URL refers to the resource to act on. Just by looking at this combination, we should already fully understand the intention of the request.

   By convention, REST uses four request methods for the following purposes:

   - POST /products - create new resource.
   - GET /products/12345 - get existing resource.
   - PUT /products/12345 - modify existing resource.
   - DELETE /products/12345 - remove existing resource.

   Notice how 'clean' URLs are used instead of the usual PHP query string? Comparing /products.php?id=12345 to its clean form /products/12345, the latter is more readable and easier to remember. It also conceals implementation details of the server which is irrelevant to the client i.e. No need to change all .php to .py on the client should you decide to migrate to python later.

   Limited as four request methods may seem, they are actually sufficient to describe any operation. Consider a request to make a purchase. Though there is no BUY method, we never append an action to the URL (POST /products/buy). Why make things more complicated when we already have a clean and built-in way of representing the operation? Instead, buying a product can be thought of as creating a 'transaction' (POST /transactions). The trick is to derive a noun from the action.

2. **Parameters**

   This is the required data to compute the results. Typically, you only need this for PUT and POST methods. Any message formats may be used here. You can consider using JSON for objects and `multipart/form-data` for large files like photos.

3. **Return value(s)**

   This is the information to return to the client after processing the request. Like the parameters, you can choose to use any message formats here. Make sure that you also plan for error responses.

If your request which can potentially return large amounts of data, consider using a paging mechanism. Imagine your Facebook news feed returning all entries since the day you joined the social network!

You may have the freedom of planning this however you like, but always try to keep your style consistent. It would also be pretty confusing if every API function uses different message formats.

There are too many REST practices to be discussed in detail here, make sure you do some research before you continue. You can find many articles and discussions online regarding RESTful API design. Note that this topic is highly debated, with many differing opinions on what constitutes REST. It is not necessary to go to great lengths just to follow every single rule, but you must be able to justify your decisions with regards to your application's needs. It is also recommended that you take a look at Facebook's Graph API[1], which conforms rather well to REST principles. In particular, observe their choice of HTTP request methods, how their relative URLs are structured as well as parameters passed.

Milestone 5: Design and document (at least 3) most prominent requests of your REST API. The documentation should describe the requests in terms of the triplet mentioned above. Do provide us with a brief explanation on the purpose of each request for reference. Also, explain how your API conforms to the REST principles and why you have chosen to ignore certain practices(if any).

Phase 2: REST Server

Now that you are done with the planning, it is time to turn your design into a usable product. In this section, mini tutorials will be provided to get you started on implementing your REST server in PHP. Since the client is probably required to test much of the server's functionality, you are advised to split the work well among your team-mates to develop the server and at least the part of the client that communicates with it concurrently.

Getting Started

The following is a simple "Hello World" REST service provider for a very minimal social networking site. This should take you less than 5 minutes to complete.

```
<?php
  $data = array(
    'id' => '0',
    'name' => 'Bob',
    'friends' => array(
      array( 'id' => '1', 'name' => 'Andy' ),
      array( 'id' => '2', 'name' => 'John' )
    )
  );
```

[1] https://developers.facebook.com/docs/reference/api/

```
  header( 'Content-Type: application/json' );
  echo json_encode( $data );
?>
```

Upload the newly-created file to your AWS instance and navigate to the URL in your browser. You should see the following text:

```
{
  "id": "0",
  "name": "Bob",
  "friends":
  [{
    "id":"1",
    "name":"Andy"
  },
  {
    "id":"2",
    "name":"John"
  }]
}
```

Congratulations, you have just built your first REST service provider :-).

The JSON extension, which ships with PHP 5.2.0 or later, provides us with functions that convert data between a PHP array and a JSON string. In the above example, we see that json_encode converted our array with Bob's data into a JSON string. It works on both indexed or associative arrays, and even objects as well. More information and examples at: `http://sg.php.net/manual/en/book.json.php`

Database Queries

We have seen that dealing with JSON in PHP is trivial using the built-in functions. However, any decent web application should do more than returning a hard-coded array. As the built-in functions do not support the query resource directly, the rows have to be fetched individually and added to an array.

```
<?php
  $res = mysql_query( "SELECT ..." );

  while ( $row = mysql_fetch_assoc( $res ) ) {
    $array[] = $row;
  }

  header( 'Content-Type: application/json' );
  echo json_encode( $array );
?>
```

HTTP Request Methods

Now that we can use the results from queries, we can move on to providing the actual API for the client to use. We can do so by handling the four HTTP request methods used in REST services: GET, PUT, POST and DELETE. As mentioned earlier, GET retrieves a resource, PUT modifies it, POST creates a new resource and last but not least, DELETE removes the resource. If it is not intended for the client to perform any of these actions, simply ignore the corresponding request method.

In the next example, consider an online store portal used by individuals who want a quick and painless way to setup a blog-shop. The portal also has an application to help store owners manage their blog-shop on the move. Through the use of a REST API, the application allows the shopkeeper to retrieve, create, edit and remove items from the store. Provided below is a skeleton of a PHP script to handle one type of resource, which is in our case the products stocked by the shop. Likely database queries and API responses that correspond to each request method have also been suggested in the comments.

```
<?php
  switch ( $_SERVER[ 'REQUEST_METHOD' ] ) {
    case 'GET':
      /*
      Query:
      SELECT * FROM products WHERE id = '" . $_GET['id']

      Response:
      {
        "id": "12345",
        "name": "Apple",
        "price": "0.5"
      }
      */
      break;

    case 'POST':
      /*
      Query:
      INSERT INTO products (name, price)
      VALUES($_POST[ 'name' ], $_POST[ 'price' ])

      Response:
      {
        "id": "12345"
      }
      */
      break;

    case 'PUT':
      parse_str( file_get_contents( 'php://input' ) , $_PUT );
      /*
      Query:
      UPDATE products
      SET name = $_PUT[ 'name' ], price = $_PUT[ 'price' ]
      WHERE id = $_GET[ 'id' ]

      Response:
      true / false
      */
      break;

    case 'DELETE':
      parse_str( file_get_contents( 'php://input' ) , $_DELETE );
      /*
      Query:
      DELETE FROM products WHERE id = $_GET[ 'id' ]

      Response:
      true / false
      */
      break;
  }
```

```
?>
```

If there is anything in the snippet that is new to you, it is probably the extra line in the PUT and DELETE cases. `file_get_contents` is used to retrieve the HTTP request body and parse_str tokenizes the data and inserts the key value pairs in an associative array. This achieves an effect similar to the built-in `$_GET` and `$_POST` arrays. However, parse_str only works for url-encoded form data. When expecting other data types, they must be parsed with their respective functions (e.g. `json_decode` for JSON).

Some code have been omitted for clarity but do remember to program defensively i.e check variables with isset and make sure to sanitize. Also, as different operations may be expecting different types of data, you should always specify the content-type of the message so the receiving end has an idea of how to handle it. You can check the content-type of the request using $_SERVER['CONTENT_TYPE'].

If you face problems trying to get PUT and DELETE to work, chances are, they have not been enabled. Look into Apache's **httpd.conf** for the Limit and LimitExcept directives corresponding to your virtual host and include the missing methods. In case you are wondering, Apache is a generic webserver which can be used for many purposes and this file is a configuration file used to customize the server for your specific needs. The location of this file may differ for the various linux distros, but you can always consult Google.

```
<Directory "/var/www">
  AllowOverride FileInfo AuthConfig Limit Indexes
  <Limit GET POST PUT DELETE>
    Order allow,deny
    Allow from all
  </Limit>
  <LimitExcept GET POST PUT DELETE>
    Order deny,allow
    Deny from all
  </LimitExcept>
</Directory>
```

In the above example, Limit (Allow from all) allows the four methods to be accessed by any users and LimitExcept (Deny from all) denies all instructions except these four from everyone. This means that only GET, POST, PUT and DELETE are allowed on the server. Even if you did not have to manually enable PUT or DELETE, you might want to consider blocking unused methods just in case. Restart Apache after you have made the changes.

Milestone 6:	Tell us some of the more interesting queries (at least 3) in your application that requires database access. Provide the actual SQL queries you used.

URL Rewrite

When we planned the REST API earlier, 'clean' URLs were used. However, this is not possible without some additional work as we know that php parameters look like ?var1=val1&var2=val2. What we need is the rewrite module, which lets us modify the URL before Apache tries to execute the referenced script. To enable it, look into httpd.conf and make sure the following lines exist and are uncommented.

```
LoadModule rewrite_module modules/mod_rewrite.so
AccessFileName .htaccess
```

Find the Directory directive corresponding to your web folder and set AllowOverride All. Also make sure to remove MultiViews from Options as it interferes with rewrite. After you're done, restart Apache and we are ready to start rewriting.

Create a file called .htaccess in your root directory with the following lines:

```
RewriteEngine on
RewriteRule ^products(?:/([^/]+))?/?$ products.php?id=$1 [QSA,L]
```

The RewriteRule matches any relative URL with the form of products/[product-id] and rewrites it to point to the real file products.php, passing the product-id as one of the URL variables. To learn more about `mod_rewrite`, refer to the official guide at `http://httpd.apache.org/docs/2.0/misc/rewriteguide.html`.

Milestone 7: Find out and explain what [QSA,L] means. Tell us about your most interesting rewrite rule.

Phase 3: Mobile Client

With a HTML5 application, native functionality and libraries of the mobile device cannot be accessed. We are left with no choice but to rely on HTML, CSS and Javascript to style and program the application. But the good news is that your application can immediately be used by any device with a modern web browser. In this section, you will learn all you need to know about creating a simple HTML5 client that works offline as opposed to normal web pages and is able to communicate with the REST server.

Getting Started

As you would have already learnt about HTML5 from the workshop, only new ideas which apply to mobile devices will be discussed here. Apart from your usual HTML5 features, most mobile devices allow users to add a shortcut of a web page to their home screen.

In iOS devices, these shortcuts will run in a standalone browser window without the browser's UI controls. This entitles web applications added to home screen to more space to work with and look as if they were native applications, in addition to ease of access. Special tags have been specified to allow developers to set icons, splash screens and even the colour of the device's status bar. Much as the users may like your application, adding a blank icon will make their screen look dull. If users are willing

to shortcut your application, do them a favour by preserving the aesthetics of their home screen. The Apple documentation tells us how to do so: http://developer.apple.com/library/safari/documentation/appleapplications/reference/safariwebcontent/ConfiguringWebApplications.html If you plan to release your application, it is a good idea to do the same for Android and BlackBerry devices.

Milestone 8:	Create an attractive icon and splash screen for your application. Try adding your application to the home screen to make sure that they are working properly.

Presentation

Your team (or maybe just your user interface designer) should spend some time designing a good UI. A good UI is the key factor that attracts users to your application. Although the functionality of your application is important, the way that it provides the functionality is just as important. An application that is difficult to use won't be used. Period. It won't matter how technically superior your application is or what functionality it provides. If your users don't like it, they simply won't use it. Seriously, do spend some time getting it right. In most cases, you'll know immediately if your UI makes or breaks it. It's common sense(!).

According to web standards, you should use CSS to style your webpage. Since the mobile client is also a webpage, the same principles apply here. CSS stands for Cascading Style Sheets, a language to determine the formatting and layout of a web page. It provides a way to separate content from presentation, thus helping us to organize the code in a much more efficient manner.

All your styling should be contained within CSS files (you can link external files to your html page) or clearly defined at the beginning of your page within the style tag. It is a very bad practice to mix CSS into HTML code.

Milestone 9:	Style different UI components within the application using CSS in a structured way (i.e. marks will be deducted if you submit messy code). Explain why your UI design is the best possible UI for your application.

Offline operability and data persistency

Very often, it is a good idea to provide the user with some level of offline operability. Even without an internet connection, your application should neither crash nor be totally useless. If there is something that can obviously be done offline, users will expect to be able to do so.

For example, photo sharing applications should allow users to organize and edit the photos anytime. The jobs can be saved in a queue if the application is not connected and uploading can commence when the application goes online.

Another example is the Dropbox iOS application. An internet connection is required to retrieve files that the user has stored in the online folder. However, it also allows users to flag selected files, which the application caches locally in the phone's memory, making them available at all times.

We shall now look at the additions to HTML5 that will enable you to realise this for your application.

1. Application Cache

For your application to work offline, the browser will need to have all files related to your application. When the browser first loads the page of your client, it will refer to a file called the cache manifest. The cache manifest contains a list of resources that the browser will retain for offline usage.

To get started, we first need to tell Apache about our cache manifest. Create a file named .htaccess in the root of your web directory and add the following line.

```
AddType text/cache-manifest .manifest
```

This makes sure that files with the extension .manifest are served with the Content-Type of `text/cache-manifest`. This is necessary as it hints to the browser the format of the file and how it should be dealt with.

Next, we need to add the manifest attribute to the html tags in all html pages belonging to the client:

```
<!DOCTYPE html>
<html manifest="cache.manifest">
  ...
</html>
```

This will point the browser to the cache manifest we are about to create. It will contain a list of resources required for your application to work offline. Resources include html, images, stylesheets as well as javascript files. Finally we proceed to create the following file...

cache.manifest:

```
CACHE MANIFEST
# version 1.0

index.html
img/icon.png
css/default.css
js/jquery-1.6.1.min.js

NETWORK:
*
```

Some points to note about the cache manifest...

- # denotes a single-line comment.
- The NETWORK section is a list of white-listed pages. If your application will request files from other domains besides your own, they need to be listed under this section or the requests will fail. * means that any domains can be accessed from your application.
- The browser will only update its local copy of the application if a change is detected in the cache manifest, a previous visitor who already has the client cached will not request for an updated client if changes were made to any files besides the cache manifest. Therefore, it is a common practice to add a comment containing a version or revision number to be incremented whenever changes to the application are made, forcing the browser to retrieve all resources.

Note that the cache manifest does not strictly need to be named as such. Make the necessary changes to the AddType directive and manifest attribute should you wish to use a different name and file extension.

Read about more features of the Application Cache at: `http://www.w3.org/TR/html5/offline.html`

2. Web Storage & Web SQL Database

The Application Cache allows resources to be retained locally, but the JavaScript variables do not survive past the lifetime of the application. When the application is restarted, it is reset to a clean state. Prior to HTML5, cookies have been used to retain the client state. But this method imposes more burden on the network as cookies are sent with every request, and neither will it work without an internet connection. Web Storage and Web SQL Database are offline storage that you can tell the browser to store data in, and they will be retained across sessions. Both APIs are accessible via JavaScript.

Web Storage is a key/value store and can be accessed through localStorage:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      var state = localStorage.getItem( 'state' );
      if ( state ) {
        console.log( JSON.parse( state ) );
        //localStorage.clear();
        //localStorage.removeItem( 'state' );
      }
      else {
        var newState = { id: '0', name: 'Bob' };
        localStorage.setItem( 'state', JSON.stringify( newState ) );
      }
    </script>
  </head>
  <body></body>
</html>
```

In the above example, nothing is observed when the user first loaded the page, but we input a new state containing the user id as well as his name into the key/value store. When the page loads for the second time, we will see that the state has already been set and will show up in the console. We may also clear the localStorage or specifically remove a key/value pair when we need to. As the storage does not support objects, a workaround is to use what we already know about JSON to do so.

Web SQL Database is a relational database which supports SQL queries.

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      var db = openDatabase( 'cs3216', '1.0', 'A sample Web SQL Database!', 5 * 1024 * 1024 );

      db.transaction( function( tx ) {
        tx.executeSql( 'CREATE TABLE friends (id unique, name)' );

        tx.executeSql( 'INSERT INTO friends (id, name) VALUES (?, ?)', [ '1', 'Andy' ] );
        tx.executeSql( 'INSERT INTO friends (id, name) VALUES (?, ?)', [ '2', 'John' ] );
      });

      db.readTransaction( function( tx ) {
        tx.executeSql( 'SELECT * FROM friends', [], function( tx, result ) {
          // handle success
          var str = '';
          for (i = 0; i < result.rows.length; i++) {
            var row = result.rows.item( i );
            str += row.id + ': ' + row.name + '\n';
          }
```

```
            alert( str );
          }, function( tx, error ) {
            // handle failure
          });
      });
    </script>
  </head>
  <body></body>
</html>
```

A database must first be opened with openDatabase with name, version, description and predicted max size. The transaction (read/write) or readTransaction (read-only) methods are used to acquire a lock on the database before queries can be executed.

For a complete list of what Web Storage and Web SQL Database can do, visit `http://dev.w3.org/html5/webstorage/` and `http://www.w3.org/TR/webdatabase/` respectively.

> **Milestone 10:** Implement and explain briefly the offline functionality of your application. State if you have used Web Storage, Web SQL Database or both in your application. Explain your choice.

3. Online/Offline Events

Storing data is easy, the tricky part comes when having to deal with syncing of states between the client and server. How does the server update a client when it connects with outdated data? How will a client post outstanding jobs when it goes online? What happens if both cases occur at the same time with conflicting instructions (e.g. client tries to comment on a thread that has already been deleted from the server)? These are but a few out of many possibilities that have to be considered.

How you handle the problem depends on your application, but this is how you can detect if your application has an internet connection:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      if ( navigator.onLine ) {
        alert( 'Online' );
      } else {
        alert( 'Offline' );
      }

      window.addEventListener( 'online', function( event ) {
        alert( 'Online' );
      }, false);

      window.addEventListener( 'offline', function( event ) {
        alert( 'Offline' );
      }, false);
    </script>
  </head>
  <body></body>
</html>
```

navigator.onLine is a boolean with the current state of the connection. You can also register callbacks for the online and offline events. Try visiting the page on your mobile device and watch the events fire as you toggle your wifi on and off.

> **Milestone 11:** Explain how you will keep your client in sync with the server. Elaborate on the cases you have taken into consideration and how it will be handled.

Communicating with the server

While offline functionality is important, not being able to communicate with the server defeats the purpose of a cloud application! Since our application runs in the browser, using an AJAX call is one of the few ways to do so. AJAX allows you to make HTTP requests using JavaScript without the need to reload the entire page. The following example uses jQuery to make a request to the REST server we have created earlier.

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.6.1.min.js"></script>

    <script>
      $.ajax({
        type: 'PUT',
        url: 'products/12345',
        //dataType: 'json',
        contentType: 'application/x-www-form-urlencoded',
        data: { name: 'Apple', price: '0.5' },
        success: function( response ) {
          // Succeeded! Do something...
          console.log( response );
        }
      });
    </script>
  </head>
  <body></body>
</html>
```

Within the AJAX call, the type refers to the HTTP request method. Just by looking at the parameters, we should already know that PUT /products/12345 is a request to modify the information of product with id = 12345. Therefore, it makes sense to send the new information to the server through the data parameter. Following Facebook's convention, a true or false response is returned for requests that do not require data. However, if you are expecting JSON to be returned, set dataType to json and jQuery will decode it for you. Also, the contentType is by default urlencoded but remember to specify this for other data types so the server knows what to do with the data. Finally, callbacks can be used to perform actions when the AJAX call succeeds or fails. Make sure that users are notified of a failure instead of being kept in suspense. Visit `http://api.jquery.com/jQuery.ajax/` for a full list of options that $.ajax() has to offer.

Authentication

Being able to talk to the server is not enough. As anyone can form and send a request to the server, you need to protect important API calls so that only authorized people can use them (e.g. only the blog owner should be able to delete his own entries). To do so, we need some method to check if the user is actually authorized to make a certain request.

The simplest approach taken by many services is to use the inbuilt HTTP basic access authentication.

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.6.1.min.js"></script>

    <script>
      $.ajax({
        type: 'DELETE',
        url: 'products/12345',
        headers: { 'Authorization': 'Basic ' + window.btoa( 'Username:Password' ) },
        success: function( response ) {
          // Do something...
        }
      });
    </script>
  </head>
  <body></body>
</html>
```

The headers key tells the ajax function to append headers to the HTTP request. windows.btoa performs a base64 encode on a username and password, which are to be separated by a colon according to the specifications. This results in the following header being sent with the request in this case.

```
Authorization: Basic VXNlcm5hbWU6UGFzc3dvcmQ=
```

The Authorization header is automatically decoded and passed to the reserved PHP variable $_SERVER, which you can then perform authentication with.

```
<?php
  if ( isset( $_SERVER[ 'PHP_AUTH_USER' ] ) && isset( $_SERVER[ 'PHP_AUTH_PW' ] ) ) {
    // Check PHP_AUTH_USER and PHP_AUTH_PW against database
  }
?>
```

The base64-encoded string may look cryptic but it is in fact reversible, which means that we are effectively transmitting a password in clear. You should at the very least apply a hash function, although using SSL would be ideal. Once again, the choice of authentication scheme for REST API is subject to much debate. But as mentioned, there are no hard-and-fast rules for designing an API. It is your job to identify the most practical choice for the requirements of your application. Basic access authentication should be sufficient in most cases, but feel free to use any methods you deem fit.

Milestone 12: Compare the pros and cons of basic access authentication to other schemes such as using cookies, digest access authentication or even OAuth. Justify your choice of authentication scheme.

User experience

Another important part of your application is user experience. Please note that user experience (usually addressed as UX or UE) is different from user interface (UI). A good UI does not guarantee a good UX at all. Sometimes, a cool-looking UI can be a disaster because of poor UX.

User experience encompasses all aspects of the end-user's interaction with the application. The first requirement for a good user experience is that the application allows the users to do what he wants with minimal fuss. Next, comes simplicity and elegance which will make the application a joy to use. User experience goes far beyond giving user what they say they want, or providing a checklist of features. In order to achieve high-quality user experience in your application, there must be seamless integration of its functionality, interaction design and interface design.

User experience is not just the job of the UI designer. Just like a good UI, you will know if an application's UX makes or breaks. It is again, just common sense. Any team member can contribute to your UX design by using it and provide feedback and suggestions. Ask your friends to use it as well to gather more feedback and ideas.

Milestone 13: Describe 1-3 user interactions within the application and explain why those interactions help make it better.

Google Analytics

Just like the Facebook application, you might be interested in the usage statistics of your application. While Facebook applications have access to Facebook Insight, which provides a lot more information about your application, the only mechanism you have in application is to embed Google Analytics (or other tracking mechanism).

Google Analytics can be set up really easily. With just a few javascript calls, you get access to detailed statistics and beautiful graphs all for free! They can give you hints about common user behaviour which you can use to to improve your application with.

Pageview Tracking can be used to track the frequency of access of individual html pages.

```
var _gaq = _gaq || [];
_gaq.push(['_setAccount', 'UA-XXXXXXXX-X']);
_gaq.push(['_trackPageview']);

(function() {
  var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
  ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-analytics.com/ga.js';
  var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
})();
```

If your application's interface is predominantly built with a JavaScript framework, it is possible that you only have one html page with all transitions handled by JavaScript code. Pages can still be tracked using Virtual Pageviews by manually entering the path for each page as the second parameter.

```
_gaq.push(['_trackPageview', '/options']);
```

Event Tracking can provide more fine-grained control over which actions to track. They can be used to track more events such as successful AJAX calls, buttons pressed, or even videos downloaded.

The following code can be embedded in the click callback of a button which publishes a new blog post:

```
_gaq.push(['_trackEvent', 'Buttons', 'Click', 'Publish Entry']);
```

To learn how to track social network activity, check out the section on Social Integration Tracking. Finally, if you remember about the cache manifest we had to create earlier, you may realise that there is one more step to be done. If the NETWORK section is not already set to allow all domains (*), we will need to manually include the two resources that Google Analytics requires.

```
NETWORK:
http://www.google-analytics.com/ga.js
http://www.google-analytics.com/__utm.gif
```

Files under the NETWORK section are whitelisted and not cached, which is appropriate for code that does tracking. ga.js contains code which reports statistics of the application. Google Analytics works by making a GET request to a 1 * 1 pixel image (__utm.gif) when something needs to be tracked. Therefore, it needs to be whitelisted so the request hits the network every single time, since fetching from the local cache will not update Google's servers with the tracking data.

Google Analytics only updates the statistics once a day, do not expect to see immediate results. There are however, signs to check that your application is being tracked:

1. Check the Network tab in the developer tools provided by your browser. Every time a tracked page or event occurs, a new GET request to __utm.gif should appear.
2. Instead of /ga.js, use the debug version available at /u/ga_debug.js, which will output logs in the JavaScript console.

More information and examples at http://code.google.com/apis/analytics/docs/tracking/home.html

Milestone 14: Embed Google Analytics in your application and give us a screenshot of the report. Make sure you embed the tracker at least 48 hours before submission deadline as updates are reported once per day.

Phase 4: Coolness Factor

This section is purely optional. Completing milestone(s) described in this section may contribute to the 30% coolness factor.

Several suggestions have been provided. We emphasize that these are merely suggestions, which means that if you find them unsuitable for your application, you may still score full points in coolness by coming up with ideas of your own. On the other hand, blindly using these suggested technologies to create redundant features will not get you bonus marks. It is about using them in creative ways to make your application more desirable to use.

Social Integration

According to statistics, more than 750 million users actively visit Facebook, 250 million of whom access the site from their mobile devices. Twitter has 280 million users, with 26 million on mobile. Where else can you find the potential to reach so many people without a single cent spent on advertisement?

Share or Retweet buttons creates a wall post or tweet with a single click of the mouse, extending your reach to friends and followers of your current users. Having just one friend per user (out of hundreds) sign up would double your size.

Integrating Facebook Social Plugins or Twitter @Anywhere can allow people to register or login with your service quickly. Helping potential users overcome the inertia of a tedious sign-up process can potentially make them less reluctant to try your application.

Best of all, most of these features can be included by copying and pasting provided code snippets or calling a few functions in the JavaScript SDK, made so simple that it is a waste to not do so. Without much additional effort, you can get your application seen and possibly used by many more people. Of course, your application must be good or no one would share it.

Milestone 15:	Identify and integrate any social network(s) with users belonging to your target group. State the social plugins you have used. Explain your choice of social network(s) and plugins.

Geolocation

Geolocation services are becoming increasingly popular with people. Many services have made their applications location-aware to provide users with more relevant results. Chalkboard for example, is a location-aware service which shows the user a list of available promotions within 1km from his location.

Fortunately, geolocation is specified in HTML5 so non-native applications can also make use of this powerful feature. The Geolocation API consists of three simple functions you can use to make your application location-aware. When calling these functions, callbacks are passed to handle success and failure cases.

1. getCurrentPosition: Invokes callback with user's current position passed as the first parameter.
2. watchPosition: Does the same as getCurrentPosition. In addition, it starts monitoring user's position and invokes callback whenever it changes. The new position is passed to the callback as the first parameter.
3. clearWatch: Stop monitoring user's position.

This code will continue to output the user's position as he walks about with his mobile device:

```
if ( navigator.geolocation ) { // if browser supports geolocation
  navigator.geolocation.watchPosition(
    function( position ) { // success
      console.log( 'Position: [', position.coords.latitude, ', ', position.coords.longitude, ']' );
    },
```

```
    function( error ) {
      switch( error.code ) {
        case error.PERMISSION_DENIED:
          console.log( 'Permission denied' );
          break;

        case error.POSITION_UNAVAILABLE:
          console.log( 'Position unavailable' );
          break;

        case error.TIMEOUT:
          console.log( 'Timeout' );
          break;
      }
    });
}
```

Showing the user his coordinates may hardly be of any use to him at all. You could take it up a level by plotting it on a map. The next example does the same thing but plots the position using Google Maps:

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://maps.google.com/maps/api/js?sensor=true"></script>

    <!-- Put this in a separate file! -->
    <style>
      body, html, #map {
        margin: 0;
        padding: 0;
        height: 100%;
      }
    </style>

    <script>
      function init() {
        var marker;
        var map = new google.maps.Map(
          document.getElementById( 'map' ),
          { zoom: 15, mapTypeId: google.maps.MapTypeId.ROADMAP });

        if ( navigator.geolocation ) {
          navigator.geolocation.watchPosition( function( position ) {
            var lat = position.coords.latitude;
            var lng = position.coords.longitude;
            var latlng = new google.maps.LatLng( lat, lng );

            map.setCenter( latlng );

            if ( marker ) {
              marker.setPosition( latlng );
            }
            else {
              marker = new google.maps.Marker({
                map: map,
                position: latlng
              });
            }
          });
        }
      }
```

```
    </script>
  </head>
  <body onload="init()">
    <div id="map"></div>
  </body>
</html>
```

For complete documentation of Geolocation API and Google Maps, visit `http://dev.w3.org/geo/api/spec-source.html` and `http://code.google.com/apis/maps/documentation/javascript/reference.html` respectively.

Milestone 16: Make use of the Geolocation API in your application. Plot it with Bing Maps, Google Maps or Gothere Maps API or even draw out possible routes for the convenience of your user.(Optional)

Native look and feel

Although your application may not be native, it may be a good idea to style it to look and feel like one. Owners of mobile devices have probably become accustomed to the user interface provided by the operating system, they instinctively know how to use a native application when faced with one. When there seems to be more information, they try to scroll down. When they perform a swipe, they expect to see the next page. Likewise, to navigate through the application, looking for tabs at the bottom of the screen is the first thing that comes to mind. By making your application native-like, its learning curve becomes less steep. In addition, a lot of research has been done by the mobile device creators to arrive at the current interface, we are less likely to make design mistakes by replicating it.

How can we emulate a native application? As part of the goal to improve the mobile web, HTML5 and CSS3 have been loaded with many new features. These features include animations, 3D transformations, visual effects and events which can reproduce the native look and feel when used in parallel. To do so however, is time-consuming and a sizeable project on its own. Fortunately, others have done the dirty work and open-sourced frameworks and libraries you can use to easily recreate the native experience. The following are some examples you can consider:

- jQTouch - `http://jqtouch.com/`
- jQuery Mobile - `http://jquerymobile.com/`
- Sencha Touch - `http://www.sencha.com/products/touch/`

You probably know this by now, but using others' code does not come without any cost. Some have very steep learning curves, others may require you to adopt a totally different style of programming and certain ones may be hardly or badly documented. Worst of all, the code may be littered with bugs and you certainly do not have the time to be fixing someone else's problems. Be sure to do a thorough evaluation before making a decision. Also be warned that the teaching staff may be unfamiliar and thus unable to assist with problems encountered with external frameworks/libraries.

Milestone 17:	Justify your choice of framework/library by comparing it against others. Explain why the one you have chosen best fulfils your needs. (Optional)

Grading Scheme

The grading of the assignment is divided into two components: satisfying the compulsory milestones(70%) and coolness factor (30%). There are 15 compulsory milestones. Excluding Milestone 3 which is not graded; the remaining 14 are worth 5% each for a total of 70%.

This assignment is due on **24 September 2011**. You are expected to deliver the following:

1. A page or two of write-up on how the application helps to achieve the objectives of the assignment
2. Completion of all compulsory milestones and write-up explaining how we can see that the milestones were achieved.
3. A short write-up pitching your application to the teaching staff, i.e. convince us that your application is so good that it deserves all 30% of the coolness factor points. Restriction: no longer than 2 pages (A4), 12pt. font, Georgia or equivalent.

Mode of Submission

The following is the list of deliverables:

1. Source code: compress all the source files into a single archive (zip/rar/tarball), maintaining the directory structure of the source files.
2. Provide us the URL to your application, i.e. your mobile cloud application must be accessible online somewhere.
3. A short readme file containing the list of group members, including matriculation numbers, name and a description of the contributions of each member to the assignment. Make sure that your application name is clearly written in the README file. The README file should also contain all the necessary write-ups for the milestones for this assignment.

Archive all deliverables into a single archive (zip/rar/tarball) and name it assignment2-[GroupNo.]. zip/rar/tarball. Upload it to CS3216 IVLE Workbin in the Assignment 2 Submission folder.

Clarifications and questions related to this assignment may be directed to the IVLE Forum under the header ”Assignment 2: Mobile Cloud Application”.

Good luck and have fun!