Flipboard-like HTML5 Mobile App Architecture Design Specification

Author	Revision Number	Date
architect	1.0	20 April 2014

• Application Design Specification

1. Design

Our client want to develop a Hybrid mobile application built in HTML5/JS/CSS, which will be deployed to iOS and Android - Phone & Tablet sized.
This mobile application will aggregate RSS data from external sources, social media content, and internal content served via REST API, similar as Flipboard.
For internal content, client would like to seek an existing content management system that fits the requirement to reduce the development effort.

The application has been broken down into 3 sub-modules:
Aggregators Assembly: This submodule is responsible for crawling the articles from the external RSSs and the internal CMS articles and videos.
REST API Assembly: This submodule is responsible for developing the REST API.
Front-End Assembly: This submodule is responsible for the development of the mobile application which will use the REST API.

1.1 Design Decisions

1.1.1 Key Requirements

- No enterprise licensing fees (open source) (open source must be MIT or Apache licensed)

  All external components (see section 1.3.2) are either MIT, Apache or BSD License. WordPress is not, but it is confirmed by client as an acceptable choice.

- Includes a REST API baked in (If not possible, must be able to build on top of it)

  We will be using our custom REST API.

- REST API must deliver content in a way that can be easily consumed and displayed on mobile device

  REST API will deliver content in JSON format which will be used by mobile to bind data to the UI. The article content itself will not be served by the REST API, it will be loaded directly using the article link in an iframe.

- CMS must support “blog style” content

  WordPress supports blog style content.

- CMS must have some sort of way to upload data.

  WordPress does have support to upload data.

- Must be able to support up to 3000 concurrent connections

  Both the REST API and the CMS will be fronted by the nginx server which is a fairly efficient server and can handle thousands of concurrent connections by default.

  The REST API is designed to be effectively stateless and can thus scale up horizontally.

  WordPress' performance under high load is unknown. However there is nothing in its architecture that precludes it from scaling up horizontally.

  In case we do need to scale, we can use nginx load balancing configuration.
  http://nginx.org/en/docs/http/load_balancing.html

1.1.2 CMS

The basic requirement for CMS is to support blog content, have good performance and optionally have built in support for REST API.

We will be using WordPress as our CMS.

WordPress is chosen primarily because it has the best plugin for video content.
https://wordpress.org/plugins/wp-video-posts/

Another advantage to WordPress is that it has the simplest database, which aids development by being self explanatory (definitely not the case in Joomla, Drupal, Concrete5 etc)

WordPress has support for REST also (unlike Joomla and Drupal) but it is not easily extensible and thus we will use our own custom REST API over the content stored in WordPress.

WordPress does support blog content like most other CMSs.

For upload existing articles, WordPress has inbuilt support using Tools>Import tool. CSV imports can also be done using the CSV importer plugin at http://wordpress.org/plugins/wp-csv/

As far as performance is concerned, since we are writing a custom REST API over the WordPress database, the performace will be as good as any other REST API.

The following CMS were also evaluated:

1. DNN - Hard to install. Works on Windows only. Very much a corporate product where it is hard to get going without the help of their *customer support*. Resources and documentation are hard to find, so much so that I gave up after a while trying to get the uploads functionality working.

2. Apache Habari - WordPress license page mentions this as an alternative to WordPress. I tried to follow their installation steps. Did not work at all.

3. Concrete5 - I was able to install and check out the features. Most features are supported. No support for bulk upload of articles. Although there is support for bulk upload of media. Biggest disappointment is that there is no support for article categories but we can write custom code for this. Also the database is very cryptic and hard to understand. Overall, it can work but would require a steep learning curve for the assemblers and decent amount of custom coding.

4. Serendipity - WordPress license page mentions this as an alternative to WordPress. Installation is simple. All features are supported except the bulk upload feature. DB is also simple enough. The UI is not attractive (as you can see http://www.s9y.org/) but there are some themes available. Customizing themes is hard and the articles always look part of a clunky blog. However I think this can be fixed by writing custom CSS.

5. Apache Sling was considered because its content management and REST API support is very good. However it is a barebone CMS framework and hence has no inbuilt blog support. So it was not used.

6. Django CMS was also considered and was chosen as a non-GPL option, but as per client confirmation we reverted back to WordPress.

1.1.3 Videos

It is assumed that videos will be uploaded and hosted inside the WordPress CMS.
Since we will display videos using HTML5 video tag, we can use only MP4 videos. This is the only format which is commonly supported by both the Android browser and the iOS browser. See http://en.wikipedia.org/wiki/HTML5_video#Browser_support
Client's existing encoders must be used to perform the conversion to MP4 format. However this conversion is out of scope of this architecture.

1.2 Work Flow Descriptions

1.2.1 External RSS Aggregation - Work Flow Description

The workflow for external RSS aggregation is as follows:

1. A cron job will launch the RSS aggregator
2. The RSS aggregator will read all the configured RSS feeds and the last crawl timestamp for each feed.
3. The RSS aggregator will visit each feed one by one. For each feed it will:
   - Read all feed items which are published after the last crawl timestamp.
   - Save them to the article table.
   - Stop when it reaches an item that was published before the last crawl timestamp for the feed. (because it has already done so)
   - Update the last crawl timestamp to the item with the maximum timestamp.

Please note that the setup of external RSS feeds in the database will be done by an admin interface which is out of scope. This should be a one-time setup.

1.2.2 Internal CMS Articles and Videos Aggregation - Work Flow Description

The workflow for internal CMS article and video aggregation is as follows:

1. A cron job will launch the internal CMS aggregator
2. The internal CMS aggregator will search the wp_posts table for articles and videos where the post_date_gmt is more than the last internal crawl timestamp
3. The internal CMS aggregator will store the article/video information in the article table
4. Update the last internal crawl timestamp to the item with the maximum timestamp.
5. Also check if any of the already crawled posts/videos have since been deleted in WordPress, and if so, delete them in our database as well.

1.2.3 App Display - Work Flow Description

The workflow for the mobile application display is as follows:

1. The user opens the mobile application which is an HTML5 application with the use of Bootstrap for responsive UI and jQuery for DOM, XHR and general JavaScript manipulation.
2. The mobile application will initially load the configurations from the localStorage of the browser.
3. The mobile application will load data using the REST API and bind it to the UI.
4. For displaying of article, the application will load the article URL in an iframe. For displaying videos, the application will load the video URL in the HTML video tag.
5. All interaction with social networks will happen straight from the application without any REST calls.

1.3 Component Requirements

1.3.1 TopCoder Software Components

None

1.3.2 Third Party Components

PhoneGap 3.4 - http://phonegap.com/

Bootstrap 3.1.1 - http://getbootstrap.com/
jQuery 2.0 - http://jquery.com/
Spring Framework 4.0.3 - http://projects.spring.io/spring-framework/
WordPress CMS 3.9 - https://wordpress.org/
MySQL 5.6 - http://dev.mysql.com/downloads/mysql/

1.4 Application Management

1.4.1 REST services

REST Services will be implemented using Spring MVC Controllers. Spring MVC already has support for Jackson for conversion of Java objects to JSON.

The REST services are described in detail in the REST_API.doc document.

1.4.2 Authentication

The REST API login method will be used to perform login into Mnemosyne.
Internally this method will call the Mnemosyne Login (which is out of scope). It is assumed however that the Mnemosyne login will return a userId, sessionId and sessionExpiresAt fields in JSON response. A custom wrapper will have to be likely written over the customer's current login process to return these fields.
The method will then save the login details to our mnemosyne_external DB.

The REST API login method will return the sessionId which the client must then store for further calls. The client must then send the SessionId HTTP header for all further calls to REST API. In case SessionId expires the REST API returns 419 Authentication Timeout and then the client must re-authenticate.

For the CMS REST API, there are no authentication requirements. The client can just call the REST API to get the content.

1.4.3 Configuration

For the mobile app, we will use localStorage to store the configuration. See http://docs.phonegap.com/en/3.4.0/cordova_storage_storage.md.html#localStorage.
The configuration values will be written always on the deviceready event.
The following items are configurable:

Parameter	Description
facebook_app_id	The application id used for Facebook Javascript SDK based Oauth login.
twitter_oauth_consumer_key	The consumer key to be used for the Twitter Oauth process.
linkedin_oauth_consumer_key	The consumer key to be used for the LinkedIn Oauth process.
max_client_log_records	If client logging is enabled, then the maximum number of log records to save. Integer value. Defaults to 100
perform_client_logging	Whether client logging is enabled. Boolean value. Defaults to false.

For the REST API, configuration will be stored in the application context xml file. All configurations, whether beans or primitives, will be injected into the controllers using dependency injection. Beans that are configurable will use @Component attribute and fields that are configurable will have a setter function with the @Autowired attribute.

Parameter	Description
relatedArticleService	The implementation of RelatedArticleService to be injected into MnemosyneRestApiController
mnemosyneRestApiPersistence	The MnemosyneRestApiPersistence object to be injected into MnemosyneRestApiController
dataSource	The DataSource to be used for accessing persistence.
numberOfArticles	The max number of articles to return for the RelatedArticleService. Must be positive integer.
recencyinDays	The recency of articles to return for the RelatedArticleService. Must be positive integer.

For the aggregators, the configuration will be stored in a properties file. The configuration will be for the dataSources pointing to the WordPress MySQL database and to our own database. The properties file configuration will be read in the main method. An example of the configuration is as follows:

```
wordpress_jdbc_driver_class=com.mysql.jdbc.Driver
wordpress_db_url=jdbc:mysql://<wp_server_ip_address>:3306/wordpress
wordpress_db_user=wp_admin_user
wordpress_db_password=password
mnemosyne_jdbc_driver_class=com.mysql.jdbc.Driver
mnemosyne_db_url=jdbc:mysql://<our_server_ip_address>:3306/mnemosyne
mnemosyne_db_user=some_user
mnemosyne_db_password=password
```

1.4.4 Logging

For REST API, log4j must be used to perform logging. The default log level must be INFO.
All expected exceptions must be logged at INFO level.
All unexpected exceptions must be logged at ERROR level.
REST method entry and exit must be logged, along with the input parameters and return values.
Sensitive user information such password, sessionId must not be logged.

For the aggregators, log4j must be used to perform logging.
Only exceptions that occur while crawling must be logged.

For the mobile application, logging will be done using Cordova Web SQL Storage.
Only errors will be logged.
This will include REST call response with 4xx or 5xx status.
This will also include unexpected JavaScript exceptions.
The schema for the Web SQL table (called client_logs) which will contain the logs is as follows:

Column Name	Description
source_method	The JavaScript method name where the error was encountered.
timestamp	The timestamp of the error
message	The error message
data	The error related data. For failed REST calls, this

Column Name	Description
	can be the stringified JSON response. For exceptions, this can be the stringified exception object.

The logs will be rolling in nature. There should be a maximum of *max_client_log_records* records in the client_logs table. If the table already has the maximum number of records, the oldest record must be deleted prior to the new record being inserted into the table.
Whether to perform logging will be handled by the *perform_client_logging* parameter. By default, it should be false.

1.4.5 *Persistence*

REST API will access the MySQL server using Spring's JdbcTemplate class which is a wrapper over JDBC.

1.4.6 *Exception Handling Overview*

For REST API, HTTP error response codes will be used to signify errors. Please see REST_API.doc for more details.

For mobile application, simple Bootstrap modals and Bootstrap error CSS classes will be used to display errors. See http://getbootstrap.com/javascript/#modals

For aggregators, the following errors can occur when loading a feed:

- The HTTP call to a feed URL fails
- The feeds URL loads but is not valid RSS or even XML

In either case, we must simply move on to the next feed and not update the last_crawl_timestamp for the feed.
For aggregators, the following errors can occur when reading an item/entry:

- The item/entry is not valid as per the RSS or Atom Spec

In either case, we must simply move on to the next item/entry.

1.4.7 *Transactions and Concurrency*

Spring Declarative Transactions should be used by the REST Controllers for calls where more than 1 DB record is mutated. We will use org.springframework.jdbc.datasource.DataSourceTransactionManager as the transaction manager.

Transactions are not needed in the aggregators, because of the nature of the exception handling. If there are errors, we just move on to the next feed or item. There is no need for the all-or-nothing behavior provided by transactions.

1.4.8 *Threading*

There are no thread safety issues.

Spring MVC controllers are singletons and will not be changed once instantiated. They will access an ACID compliant database (MySQL), where locks will prevent any concurrency related issues that may arise from concurrent threads. Please see TCUML for thread-safety of entities and RelatedArticleServiceImpl.

On the device, we are using HTML5 and Javascript. Javascript is inherently single threaded.

1.4.9 *Internationalization*

None

1.4.10 *Caching*

Caching in the front end is currently out of scope. In the future the article headers and categories can be cached using Cordova Storage i.e. Web SQL.

1.4.11 *Risks*

There is a clear risk that the current approach of displaying articles as-is (like they would appear in a mobile browser), will not be engaging enough for the end user.
The approval to go ahead with this assumption was provided at http://apps.topcoder.com/forums/?module=Thread&threadID=816471&start=0&mc=11#1869274

The client can address this issue by using the Flipbaord approach of extracting articles from web pages using predefined templates, in a later phase. This issue is more discussed at http://apps.topcoder.com/forums/?module=Thread&threadID=816845&start=0

1.5 Deployment Constraints

WordPress CMS will be be fronted by nginx 1.5 HTTP server.

The REST API will be bundled into a war file, and then be deployed into a Tomcat 7 container. This can then be fronted by nginx 1.5 HTTP server. We will use nginx 1.5 because of its superior memory footprint and concurrent requests handling compared to Apache Server.

The aggregators will be bundled into a JAR file which will be launched periodically using a cron job.
The mobile application will be built and deployed onto production system using PhoneGap Build which provides a build process on the cloud. See https://build.phonegap.com/ and http://docs.build.phonegap.com/en_US/. For development though, an xCode IDE connected by USB to a phone/tab should suffice. See http://docs.phonegap.com/en/2.5.0/guide_getting-started_ios_index.md.html

1.5.1 *Deployment Software*

Unix – Any flavor

Tomcat 7 Container

nginx 1.5 HTTP Server – Fronts the REST API and can be used a load balancer as well

1.6 Changes to Existing System

This is a new system

2. User Interface

The mobile app user interface will be developed using PhoneGap. The interface therefore consists simply of HTML5 web pages. jQuery provides the basic JavaScript engine for DOM manipulation and XHR calls. Bootstrap provides the responsive (mobile friendly) CSS classes and some other components (like the modal)

The user interface of the app consists of square boxes with content in them. These are called **tiles**. BootStrap Grid system will be used to display these tiles. http://getbootstrap.com/css/#grid The advantage of using Bootstrap is obvious here. The grids will auto adjust themselves for different screen sizes and orientation.

Within the application, the flipping between the categories on the home page and the flipping between the articles themselves can be done using CSS3 transitions. Very good example of various styles of transitions are shown at http://tympanus.net/Development/PageTransitions/.

Please note that CSS3 transitions are supported both in Android and iOS browsers - http://caniuse.com/css-transitions

As far as the flipping between pages of an article is concerned, this is still an open item at http://apps.topcoder.com/forums/?module=Thread&threadID=816845&start=0

2.1 Component Interface

None

2.2 Prototype Conversion

None

3. Included Documentation

3.1 Architecture Documentation

- Interfaces Class Diagrams
- Application Design Specification
- Assembly Requirement Specifications
- REST API Document
- ERD Diagram
- Sequence Diagrams