

# END SEMESTER PROJECT

on

## ANALYTICS FOR METASTUDIO

by

**Pranjal Gupta**

2013B4A7470P

At

**Homi Bhabha Center for Science Education**  
TIFR, Mumbai

**A Practice School I station of**  
BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

May 22, 2015 - July 16, 2015

# I. ACKNOWLEDGEMENTS

Industrial Training strengthens our theoretical, practical and technical concepts, which enhances our skills in the field of technology. The successful realization of the project is the outgrowth of a consolidated effort of the people from disparate fronts. It is only with their support and guidance that the developer could meet the end.

I would have never succeeded in completing my training without the cooperation and encouragement provided to me by various people. Firstly, I would like to express my gratitude towards Professor Jayashree Ramadas, the head of the institution - Homi Bhabha Centre for Science Education(HBCSE) for giving me an opportunity to work for such an auspicious internship program. I would also like to thank our coordinator at the PS program Professor G. Nagarjuna for motivating us towards the projects and providing us a direction to focus upon. Further, I would like to thank our PS instructor Dr Imlak N Shaik and our co-instructor Mr. Atul for their continuous guidance and support. Moreover, I thank the research assistants Mr. Kedar, Mr. Sunny, Mr. Mrunal, Mr. Satej, Ms. Rachana and Mr Ullas for helping me to get used to working in the open source environment and helping me out with my doubts.

Last but not the least, I would also like to thank my peers at the PS station for their valuable suggestions and inputs towards the betterment of the project. Working at Homi Bhabha Centre for Science Education as a Research Intern has been an enriching experience for me and I would like to express my deep gratitude towards everyone associated with the Project.

# BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

PRACTICE SCHOOL DIVISION

**Station :** Homi Bhabha Center for Science Education Center, TIFR  
Anushakti Nagar, Mumbai

**Duration :** 8 weeks

**Start Date :** May 22, 2015

**Submission Date :** July 16, 2015

**Title of the project :** Analytics of Metastudio

**Name & ID :** Pranjal Gupta  
2013B4A7470P

**PS Faculty :** Dr. Imlak N Shaikh

**Team members :** Akshit Bhatia (2013B3A7722P)  
Himanshu Singh Dhoni (2013A7PS187P)  
Shubhi Rastogi (2013B3A7521P)

## **Key words:**

Gsystem, MetaStudio, MongoDB, Python, Django, Django-mongokit, Gstudio, RSS-feed, JSON, XML

## **Project Areas:**

Entire project is based on open source technologies. Project is based on the metastudio environment. The areas of the project are as follows:

Website development :

Frontend: HTML, CSS, Javascript -

Backend: Django, Python Database Handling - Django-mongokit, Mongokit, MongoDB  
Algorithms - Map Reduce, Stemming,

## **Abstract:**

The aim of the project is to develop analytics feature, at user and group level, and implement RSS feeds for the metastudio platform.

# TABLE OF CONTENTS

<b>SNo.</b>	<b>Title</b>
<i>I</i>	<i>Acknowledgement</i>
<i>II</i>	<i>Abstract</i>
<i>III</i>	<i>Introduction</i>
<b>1</b>	<b>The Metastudio environment</b>
1.1	The Basics
1.2	Understanding the models
<b>2.</b>	<b>Analytics for MetaStudio</b>
2.1	What is analytics
2.2	Why do we need analytics on MetaStudio
<b>3.</b>	<b>Setting up Analytics</b>
3.1	Changes to Benchmarks model
3.2	Analytics model
3.3	Normalizing raw data
3.4	App specific analysis function
3.5	Batch processing mechanism
3.6	Process workflow
<b>4.</b>	<b>Analysis on MetaStudio platform</b>
4.1	User analytics
4.2	Group analytics
<b>5.</b>	<b>Feeds</b>
<b>6.</b>	<b>AJAX</b>
<b>7.</b>	<b>Technologies Used</b>
<b>8.</b>	<b>Future scope</b>
<i>IX</i>	<i>Conclusions</i>
<i>X</i>	<i>References</i>

### III INTRODUCTION

The aim of the project was to design an analytics feature , and implement RSS feeds for the MetaStudio platform developed by Homi Bhabha Center for Science Education(HBCSE). This feature was developed from the scratch, as no work had been done before.

The entire project is based on open source technologies, following the GNU manifesto and under the Creative Commons License.

The project was completed in three phases:

**Phase 1** : To modify the “*Benchmarks*” collection to obtain data required for performing analytics, and define “*analytics\_collection*”, a mongodb collection that stores data obtained after processing the data from benchmark.

**Phase 2** : To compare the data in benchmark and analytics, and normalize the raw data from benchmark if the two data aren’t equally updated.

**Phase 3** : To use the data from analytics collection to generate user and group analytics, and implement RSS feed for the metastudio platform.

This feature would enable analytics at group and user level. Group analytics include summary of activities done in a group, recent activities, member-wise activities, while user analytics deal with the corresponding user counterpart.

MetaStudio is a collaborative platform, and analytics will help in estimating the contribution of various members. Also, this is a first step towards generating comprehensive user reports, including timeline, which can be used by course instructors to monitor individual students and optimize the course structure. RSS feeds will enable the group members to get updates on major group activities, creation, modification, and deletion of content, without the need to visit the site.

# 1. THE METASTUDIO PLATFORM

## 1.1 THE BASICS

### The MetaStudio Initiative

This is an initiative of the Homi Bhabha Centre for Science Education, TIFR , Mumbai, India for establishing collaboration among students, teachers, researchers or anyone else interested, to shape education and research. The term MetaStudio refers to any workspace where a group of people work together, plan (design), make and display a product iteratively till they reach a final result enabling learning in a situated context, unlike a detached learning that happens through reading and memorising from books.

### What is MetaStudio?

A metaStudio, then, is a studio where we can design, deliberate and construct studios for learning. Thus, it can be conceived as an academy for designing multiple learning contexts. This can be further conceived as a remedy for the existing schools and colleges where children are forced to learn abstract topics without understanding in what context they are used and learn topics that do not mean much to their lives and interests.

### Facilities Offered by this Platform

- a wiki: to create wiki style pages collaboratively on any topic or subject
- form a group: a private, public, an invite-only, visible or invisible group, send messages to members within the group
- Write about a topic inviting discussion
- Ask questions and also respond to questions asked by others
- upload useful resources such as lessons, documents, pictures, videos, articles, software etc.
- post announcement of events as well as report on them, to tell everyone in the group
- write blogs on topics of your interest

### Site Protocol: Creative Commons using open standards

As a project inspired by free software philosophy , this site encourages the members to adhere to the principles that protect freedom, sharing, collaboration and socially

acceptable protocols. When you upload resources (digital documents and software) please ensure that you are uploading them under the Creative Commons license or other copyleft license or public domain.

## 1.2 UNDERSTANDING MODELS

### Node oriented computing

- We try to model the knowledge as nodes in a network. The interpretation of a node depends on the links the node has with other nodes, and the interpretation of the other neighbouring nodes in turn depends on their network and so on.
- There are some nodes whose interpretation is implicitly available to the computing system. The network is constructed by interlinking all things to these primitive nodes.
- New nodes obtain their meaning due to their relations with the primitives. Linking unknown symbols with known symbols provides a mechanism to define new meaning from the already available meaning.

### The Data is Modeled on the following principles :

1. Everything is a node. The memory is a set of nodes with all nodes with a unique Node ID.
2. A node links with the neighborhood nodes through relations, and some predefined name and value pairs called attributes.
3. Every node is described by its neighbourhood, i.e. the set of relations and attributes a node possesses.
4. A network can be obtained by merging the neighborhood of all the nodes as a graph.
5. The set of all attributes and relations is the full set of knowledge represented in the network in the form of propositions.
6. Each attribute or relation constitutes one "triple" expressing one proposition

## **2. ANALYTICS FOR METASTUDIO**

### **2.1 WHAT IS WEB ANALYTICS ?**

Web analytics is the measurement, collection, analysis and reporting of web data for purposes of understanding and optimizing web usage.

It can be used to draw useful conclusions about usage pattern of the website by studying user behaviour.

### **2.2 WHY DO WE NEED ANALYTICS ON METASTUDIO ?**

Metastudio is a platform for collaboration and content creation. The main purpose of the platform is to carry out online courses and monitor the members contribution and progress. Hence, analytics forms the integral part of this objective. Tracking user activities can be valuable to study user engagement with the content and can serve the dual purpose of optimizing web content as well as providing User report and feedback

- MetaStudio needs analytics to keep track of user progress on website's apps like file, forum, page, courses, etc.
- Individual activities in group can be used to study and analyze user engagement with the website.
- Group analytics give an idea about the amount of activity going on in a group. This can be used to infer about how active the group is and the extent of collaboration in the group.
- Activity report can act as a tool to measure student's response to a course hosted on MetaStudio.



## 3. SETTING UP 'ANALYTICS'

Analytics for Metastudio has to be tailored from the very start, since it was a new feature to be developed on the metaStudio platform. We need to think and compare the 'how to' of different approaches that were possible for setting up analytics on the project. As the part to complete the project, we had to make a few changes to the existing model of metaStudio and add new tables and analysis method to process user activities, apart from creating views to for showing up Analytics. The entire course of setting up the 'Analytics' module can be divided into comprehensive steps :

1. Changes to the existing **Benchmarks** model.
2. Setting up the **analytics\_collection** Model to store the processed activity data.
3. Functions to normalize and filter the unprocessed benchmark data.
4. Determining the activity and writing app specific analysis functions.
5. Setting up Batch processing, wherein only the raw fresh data is processed.

The Analytics is implemented as a separate module in NDF structure. All the views and functionality is implemented in **analytics.py** in the **/views** folder, with corresponding URLs in the **/url/** folder

### 3.1 CHANGES TO BENCHMARK MODEL

The Benchmark model implemented in MetaStudio is used for the purpose of benchmarking the called functions through specific URLs. Hence, this model registered all the user calls from its clients. However, it does not register the identity of the user who made the call.

The Benchmark model was, hence, changed to hold the identity of the user who made the call to a URL. The existing database scheme was altered to store the **user**, **session\_key** and **group\_id** variables that can be used to identify the user and the group in which a particular activity is being performed. In this way each activity (that has a URL), is binded to a specific user in specific session, be it an Anonymous User or a logged in member. Likewise, the Benchmarks function in the **methods.py** was changed to insert data in the new fields.

The following are the new fields added the the existing Benchmark schema :

- **user (basestring)**  
takes in the username of the client from the request object ( request.user.username ), whenever it is available on function invocation.
- **group (basestring)**  
holds the group's name or ObjectId which is the first parameter in the URL's path since each user activity is in a group.
- **action (basestring)**  
'rough' activity of the user. It is basically the third and fourth parameter in the URL's path (whenever present).
- **session\_key (basestring)**  
It is a session variable which is unique to a particular user. Session information is available in the request object of the functions that is being directly called by a URL.
- **has\_data (basestring)**  
A boolean dictionary that holds whether the call to a function is accompanied by the POST or GET data in the request object. This is used determine the preference order of activities having same URL and hence, reduce redundancy.

Model changes:

EARLIER 'BENCHMARK' SCHEME	PRESENT 'BENCHMARK' SCHEME
<pre> {   "_id" : &lt;mongo objectId&gt;,   "_type" : "Benchmark",   "name" : "group_dashboard",   "parameters" : "1",   "size_of_parameters" : "68",   "last_update" : &lt;datetime Object&gt;,   "calling_url" : "/home/",   "function_output_length" : null,   "time_taken" : "0.932454824448" } </pre>	<pre> {   "_id" : &lt;mongo ObjectId&gt;,   "_type" : "Benchmark",   "group" : "559bb559421aa936242ff73a",   "name" : "user_summary",   "parameters" : "1",   "function_output_length" : null,   "size_of_parameters" : "148",   "last_update" : &lt;datetime Object&gt;,   "calling_url" : "/home/",   "user" : "g31pranjal",   "action" : "analytics/summary",   "session_key" : &lt;session_key&gt;,   "time_taken" : "0.128052949905",   "has_data" : {     "POST" : false,     "GET" : false   } } </pre>

## 3.2 ANALYTICS MODEL

The data in the Benchmarks collection is the raw data which have :

1. Function calls which is not invoked by a specific URL (we do not want activities without URL since, these are invoked by other functions )
2. Unwanted activities like calls to image thumbnails, own profile, home, logout etc.
3. Multiple activities having same URL wherever single URL invokes multiple methods giving rise to redundancy that have to be filtered out.

Hence, because of the following constraints the 'raw' data from the benchmarks model cannot be directly used for analysis. We need a separate collection that stores the processed 'relevant' data that can be exhaustively used for analytical purposes. Hence we define the following model :

The current general scheme of the Model is :

ANALYTICS MODEL
<pre>{   "_id" : ObjectId("55a35d59421aa91e1d8ed695"),   "session_key" : "dk5wms1kg23hn8snjhijxyryi5e2c57",   "obj" : {     "page" : {       "url" : "page",       "id" : ObjectId("558a5e13421aa9619ccb2176"),       "name" : "Hias"     }   },   "timestamp" : ISODate("2015-06-24T16:13:17.075Z"),   "user" : {     "name" : "pranjalgupta",     "id" : 2   },   "action" : {     "phrase" : "viewed a",     "key" : "view"   },   "group_id" : "55701d90421aa9176c649ee7"   "type" : "data" }</pre>

- **user (dict)**  
takes in the username and the user\_id of the logged in user.
- **action (dict)**  
It defines the details of the 'action' that the user takes on the platform. It take 2 necessary attributes
  1. **key** : one word verb that defines the activity of the user, like, **create, edit, delete, add** etc.
  2. **phrase** : the verb phrase that will appear will representing the user activity in the form of a sentence (*explained later*)

The motive of the 'key' is to sort the collection by action. Hence, this should be a generic word describing the activity.
- **obj (dict)**  
obj is the metaStudio Node object on which the 'user' performs the 'action'. This dictionary takes in one or more 'key : dict' entries wherein the '**key**' is the gApp on which the activity is performed like page, file, forum etc. while the '**dict**' is the collection of properties of the entry, like **id, link, name** etc.
- **timestamp (datetime)**  
takes in the time of the activity
- **session\_key (basestring)**  
takes the session\_key of the session in which user does the activity
- **group\_id (basestring)**  
takes the group\_id of the group in which the user does the activity.
- **type (basestring)**  
type is set to 'data' in case the activity of the user is being stored. It is set to 'log' to record the activity of user triggering Analytics module.

The Analytics model serves the purpose of storing the user activity log as well as logging the User activity on the Analytics module. The data of this model is stored in analytics\_collection, as a separate and standalone collection.

This Model implements the RDF scheme, wherein the content can be represented using semantics. Each activity in the model can be represented in the form of subject, predicate and object as -



and the semantic sentential representation is evident from the following :

```
user.name | action.phrase | (obj.keys())[0].name
```

### 3.3 NORMALIZING 'RAW' DATA

The 'raw' data from the Benchmarks model needs to be filtered to take out unwanted activities, remove redundancy, determine objects etc. This is done by a function in the analytics.py view file defined as : `def normalize(cursor, details)`

The normalize() function takes in two arguments `cursor` and `details`.

- `cursor` : It is a database cursor that contains 'raw' activities that need to be processed.
- `details` : Is a dictionary that contains info about the user whose data needs to be processed.

The processing takes place in three module :

- **FILTERING :**

The main target is to filter out unwanted activities from the list. This may include called URLs to specific targets, like `/home/`, `/file/thumbnail/`, `/` etc. All the specific targets are mentioned in the list named `filtering_list`. Other blocks in this module includes removing visits to self dashboard, all AJAX requests.

This module can be further extended to block several other URL patterns. An important aspect of this module comes to play in implementing privacy levels in future wherein according to the User's privacy settings, a generic set of URLs can be ignored for processing.

- **REMOVING REDUNDANCY :**

There are multiple entries in the `Benchmarks` model having the same URL, originating from calls to multiple functions from the same URL. This needs to be dealt with in order to avoid a single action being registered multiple time in the Analytics collection.

This is done by grouping all activities having same URL, user and session that occur over span of 5 minutes into a single cluster and then taking a single `most`

**significant activity** from that cluster. The significance of the activity depends upon whether the activity has POST and GET data.

Only the filtered activity is taken further.

- **ROUTING TO APP SPECIFIC ANALYSIS FUNCTION :**

The activities that pass through is then routed to specific functions for different apps by means of a routing table **gapp\_list**.

```
def gapp_list(gapp):
    return {
        "page": page_activity,
        "file": file_activity,
        "course": course_activity,
        "forum": forum_activity,
        "task": task_activity,
        "event": event_activity,
        "dashboard": dashbard_activity,
        "group": group_activity,
    }.get(gapp, default_activity)
```

This list can be configured to add more apps in the future.

### 3.4 APP SPECIFIC ANALYSIS FUNCTIONS

App specific function analyzes a particular activity of its kind by means of its URL to determine the exact nature of the activity and likewise, prepares the object of Analytics model type and inserts it into the collection.

At present, we worked with the page, file, forum , tasks, dashboard apps to fully determine the activities that needs to be tracked.

For analyzing a new gApp, a new function has to be added that analyzes the structure of the URL along with the info on user and REQUEST data. Since, each app-specific function has a 'pluggable' nature, writing a new function is independent of the earlier code and can be plugged by defining the route parameters in the **gapp\_list** as,

```
"forum": forum_activity,
```

where **forum** is the app name and **forum\_activity** is the name of the definition which analyzes the forum app.

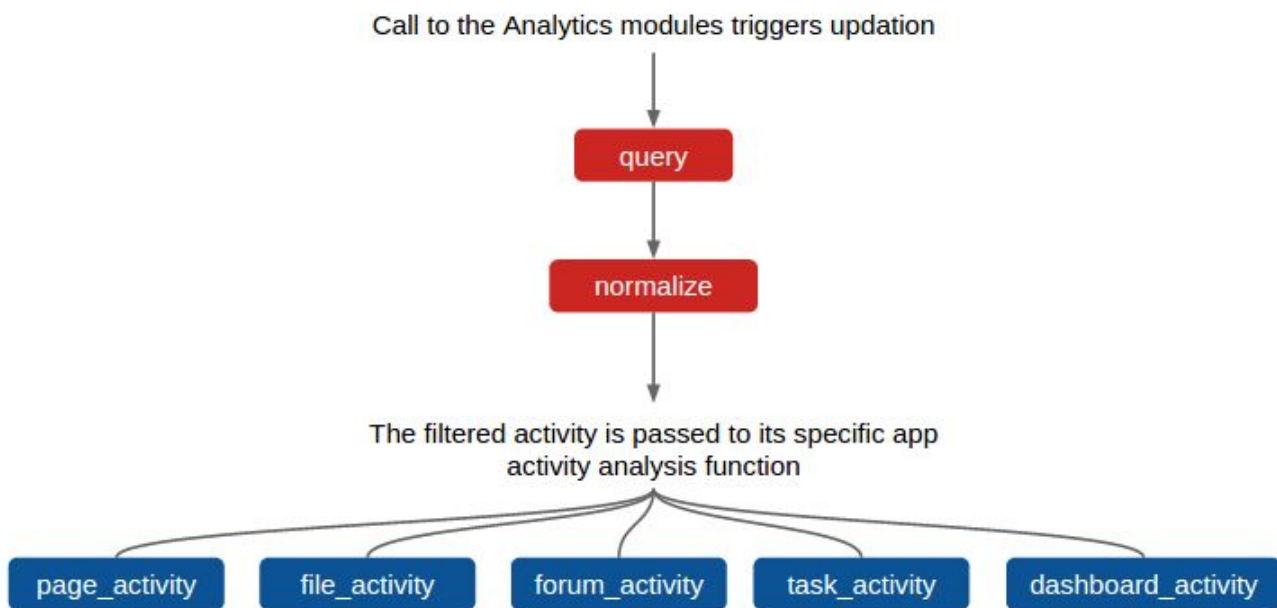
### 3.5 BATCH PROCESSING MECHANISM

Processing the entire data of a particular user from the Benchmarks collection and analyzing it, can become a hectic and costly task with the scaling of the metaStudio platform. The deployment of Batch Processing relieves the inefficiency, wherein each data is processed just once and inserted into the Analytics collection. This is achieved through a function that determines the unprocessed data : `def query(analytics_type, details)`

The function basically being triggered by the View functions of the Analytics module, is used to trigger the normalize function with the relevant data to process. this may be evident by the following flow :

1. Queries analytics\_collection type 'log' to get the last User activity on Analytics
2. Gets the unprocessed data (having timestamp greater than that available from Analytics collection)
3. Passes the raw data to `normalize(cursor, details)`
4. For Group, the contributors of the group are queried from the group NODE and data for each user is recursively updated through this procedure

### 3.6 PROCESS WORKFLOW



## 4. ANALYSIS ON METASTUDIO PLATFORM

The focus is on analyzing the data to draw useful conclusions based on the activities of user. The analytics is divided into two domains :

1. USER ANALYTICS
2. GROUP ANALYTICS

The **User Analytics** aims to analyze the activity of each user (collaborator, teacher, student) on the platform, with focus on user sessions and logging so as to detect activity behavior, by computing number of user sessions, average duration of each session, pageviews, number of pages or files created, discussions or forums he/she took part in , upvotes, ratings, etc. The parameters can then be evaluated to get the 'overall user engagement' with the platform. There will be two views in the user analytics, one will be the summarized view which will give the gist of the activities done in metaStudio whereas the other view will give the detailed activities of that user.

The **Group Analytics** aims to analyze the activities of every user present in a specific group . Most active users, most active forums, all the recent activities, contributions of each user, etc can be viewed only by the group analytics. There will be 3 views in the group Analytics,

1. It will be the summarized view of the group activities which shows the admin the overall activities and the recent activities.
2. The second view will be the detailed view of the user activities.
3. The third view will show the contribution of the members in the group.



## 4.1 USER ANALYTICS

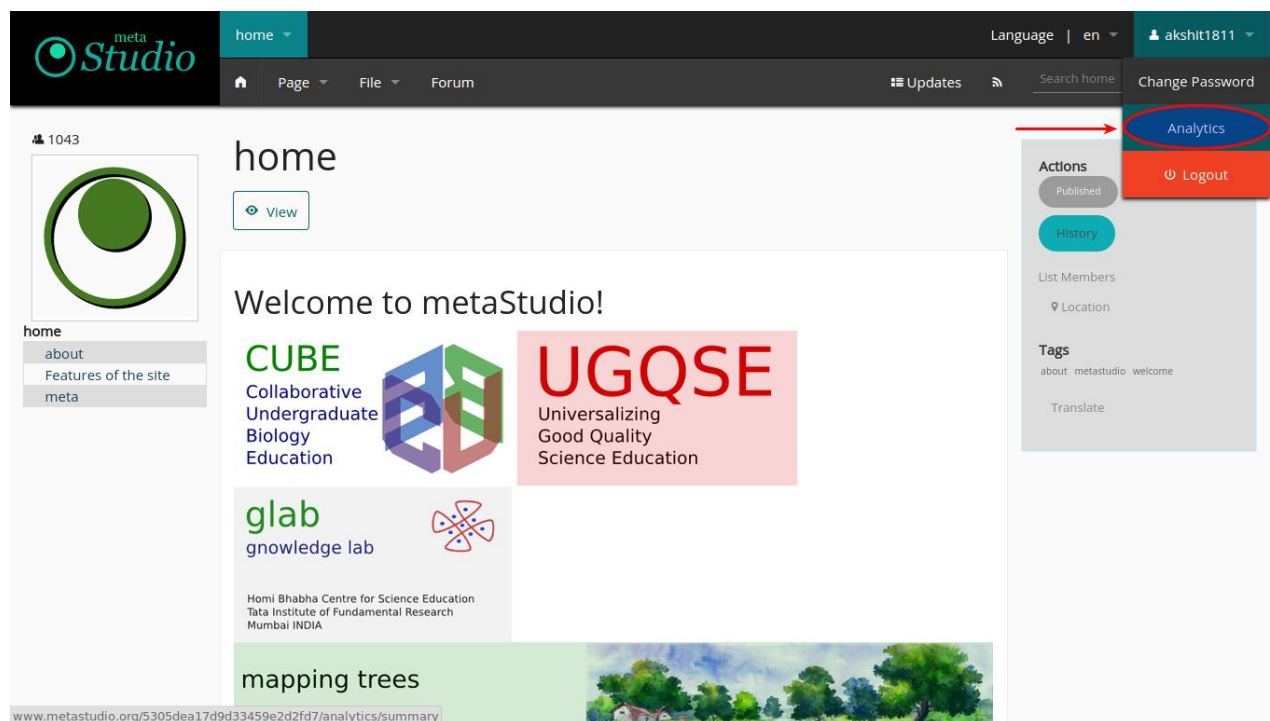
User analytics enables the user to view his summarized and detailed data of the activities done in metaStudio like the pages created, files uploaded, threads created, etc. with a link to the respective item.

The views for this were written in **analytics.py**, under “USER ANALYTICS”.

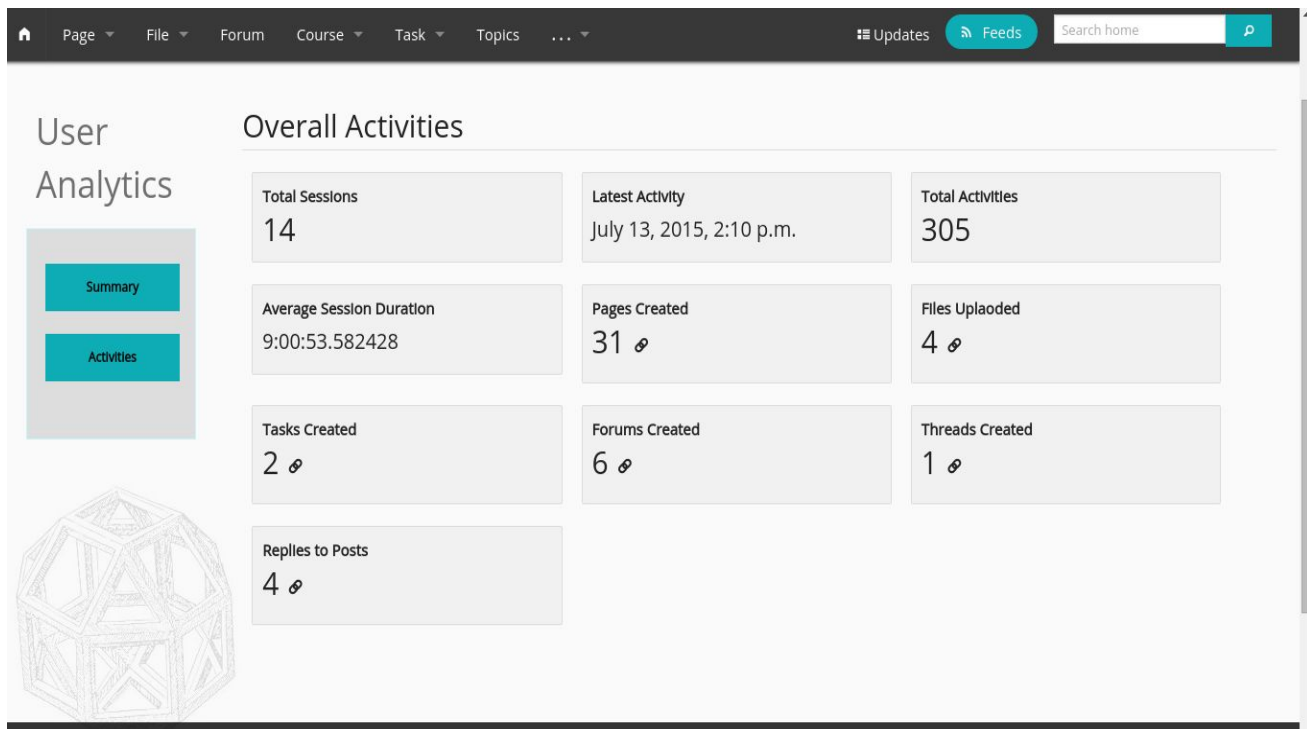
It can be viewed by clicking on analytics button as shown in the figure :

It calls the functions **user\_summary** , **get\_user\_sessions** for getting the user summary. The view “**user\_summary**” takes two arguments : **request object** and **group\_id**. The analytics\_collection is queried to give the number of sessions, forums, threads, replies, files, pages, and total\_activities. The query parameters include **user name** and the **action key**. This information is stored in a list named **data**, and rendered on the template **analytics\_summary**.

The User Analytics can be accessed from the User’s Drop-down menu that appears on the header of the base.html template. *The Analytics module is, at present, separate from the Dashboard module.*



The analytics view of **user summary** will be shown as :



The detailed activities view calls the functions **user\_list\_activities**, **get\_user\_sessions** , **user\_app\_activities** for getting the user detailed activities.

This view displays the detailed activities of the user sorted in descending order to time . Clicking on the user or the gapp will redirect you to their respective views. Former is accomplished using the **user\_app\_activities** function which takes **request**, **group\_id** and **part** as arguments. A query is run in **analytics\_collection** which returns us a cursor by which we compute the number of forums, pages, etc. created by the user. All this information is stored in **date\_col1** a dictionary which is appended to list - **lst** and rendered to the template **analytics\_list\_details**.

The detailed view of the **activities** done by the user in chronological order is shown as:

The screenshot displays the MetaStudio user analytics interface. On the left, a sidebar titled "User Analytics" contains two buttons: "Summary" and "Activities". The main content area is titled "Detailed Activities for user : akshit1811" and shows a list of activities for the date "13 Jul, 2015". The activities are listed in chronological order, with the most recent at the top. Each activity entry includes the user's name, the action performed, the time of the activity, and the time elapsed since the activity occurred.

Activity	Time	Elapsed Time
akshit1811 created a page : test6	July 13, 2015, 4:51 p.m.	0 minutes ago
akshit1811 published a page : test6	July 13, 2015, 4:51 p.m.	0 minutes ago
akshit1811 created a page : test6	July 13, 2015, 4:51 p.m.	0 minutes ago
akshit1811 viewed a forum : sd	July 13, 2015, 4:29 p.m.	22 minutes ago
akshit1811 viewed a page : sa	July 13, 2015, 4:29 p.m.	22 minutes ago
akshit1811 published a page : sa	July 13, 2015, 4:29 p.m.	22 minutes ago
akshit1811 viewed a page : akfht	July 13, 2015, 2:10 p.m.	2 hours, 40 minutes ago

11 Jul, 2015

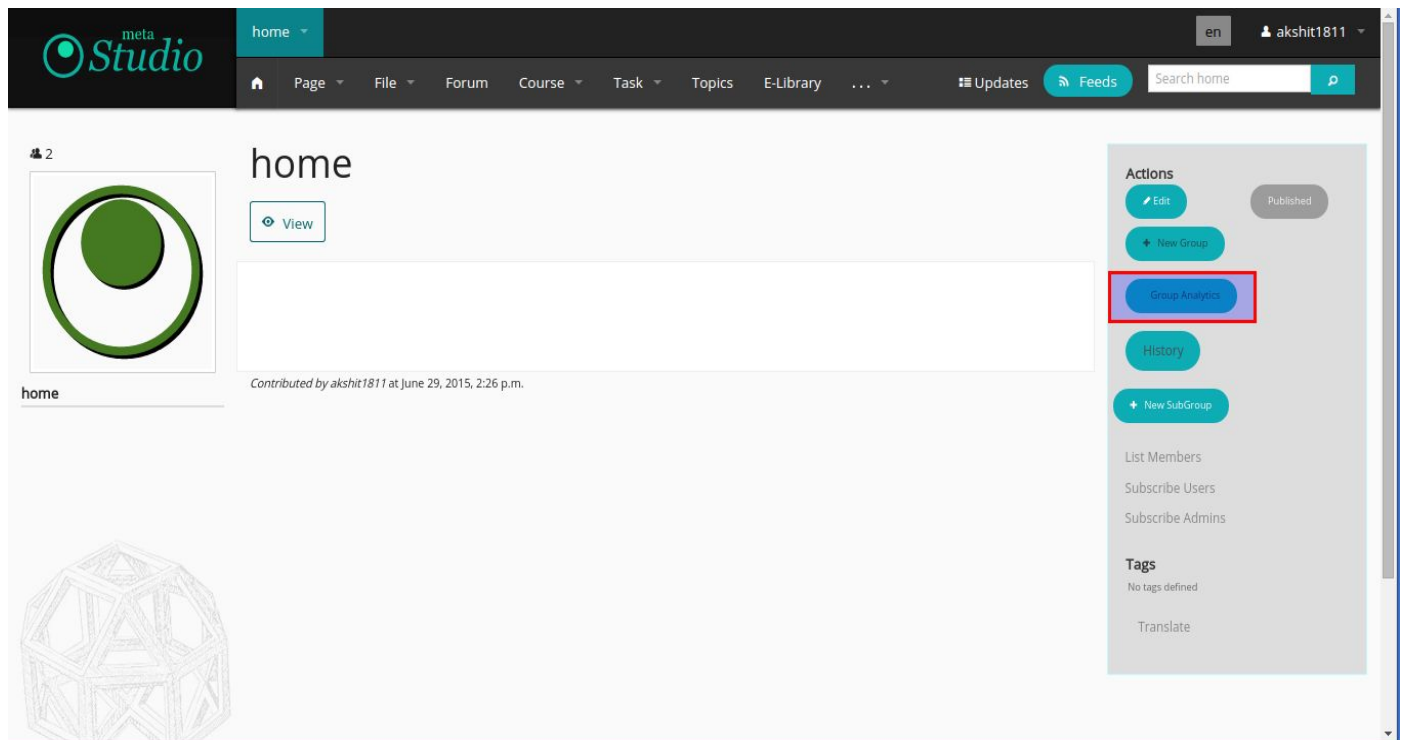
The view `user_list_activities` takes `request` and `group_id` as arguments

## 4.2 GROUP ANALYTICS

Group analytics enables the group admin to view the summarized data about the activities of group, like the number of pages, files, forums in the group, recent activities, including deletions and modification, and contributions of all members, with a link to their dashboard.

The views for this were written in **analytics.py**, under "GROUP ANALYTICS".

Group analytics can be accessed via the analytics button that appears on the group's home page, as highlighted in the figure :



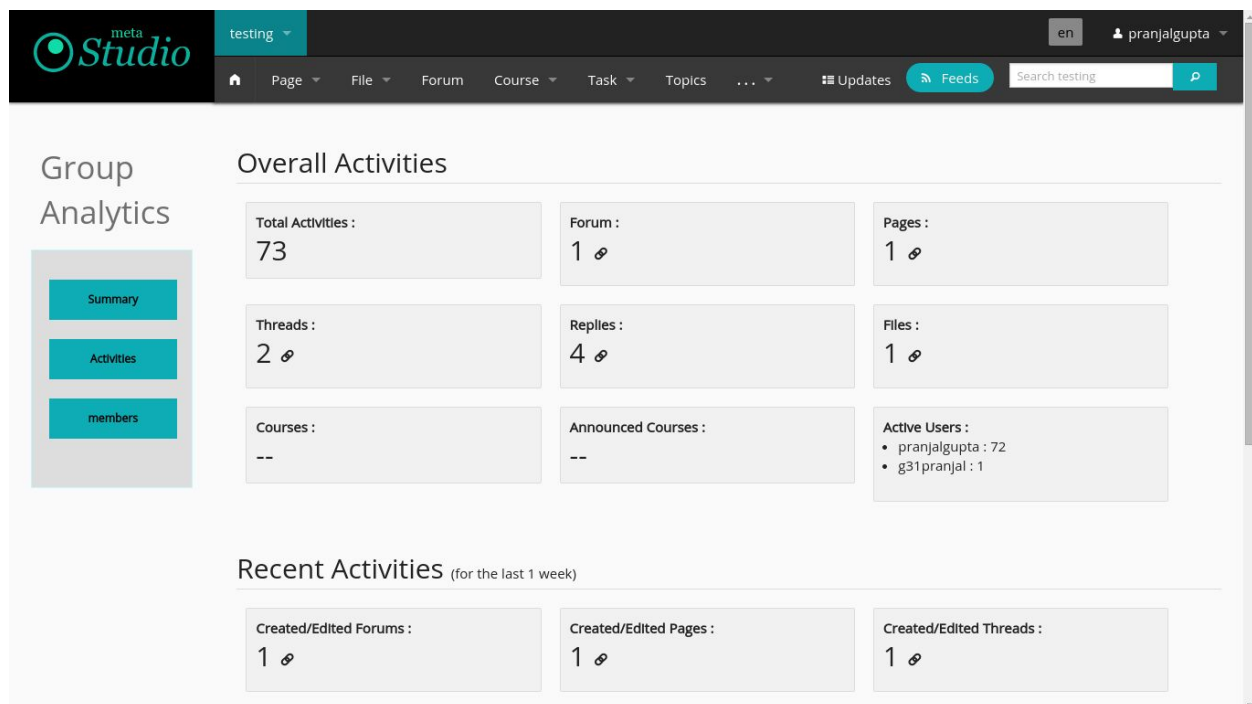
Next image is the group summary view under group analytics.

The view `group_summary` takes two arguments: `request` and `group_id`. `get_group_name_id` function is called that returns `group_id` and `group_name`.

**active\_users:** Mapreduce is used to determine the number of active users. A variable pipe is defined, which is actually the map-reduce function. `$match` retrieves all the

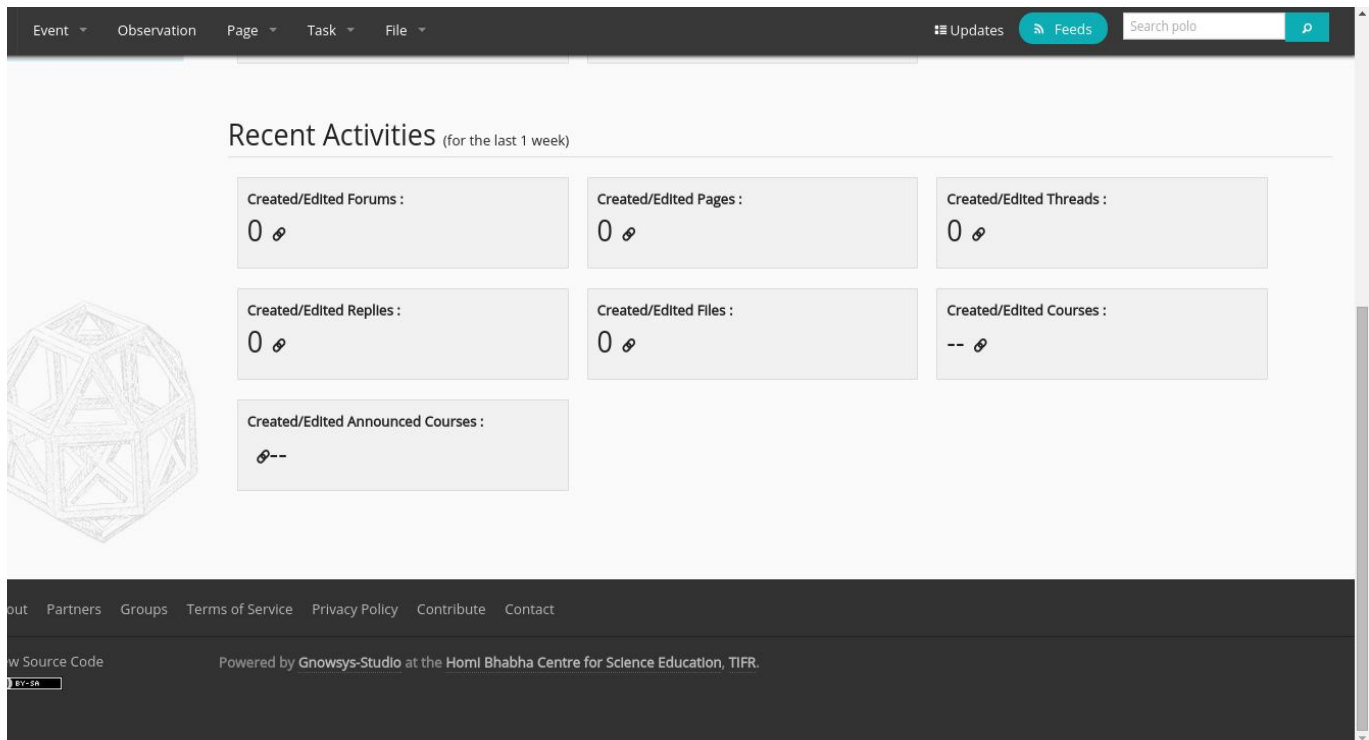
documents with the given **group\_id**. **\$group** groups the matched documents according to **user\_id**, and returns a new doc with fields **\_id** defined as **user.name**, and **num\_of\_activities** defined as the number of documents of the particular user. This is passed as a value to **pipeline** in the map-reduce operation, which returns a cursor.

**number of forums, threads, replies, files, pages, and total\_activities:** Nodes collection is queried to determine these fields. Query parameters include **url** (the app name), **group\_set**, and **status** (it should be DRAFT or PUBLISHED). This information is stored in a list named **data**, and sent to the template **analytics\_group\_summary**.

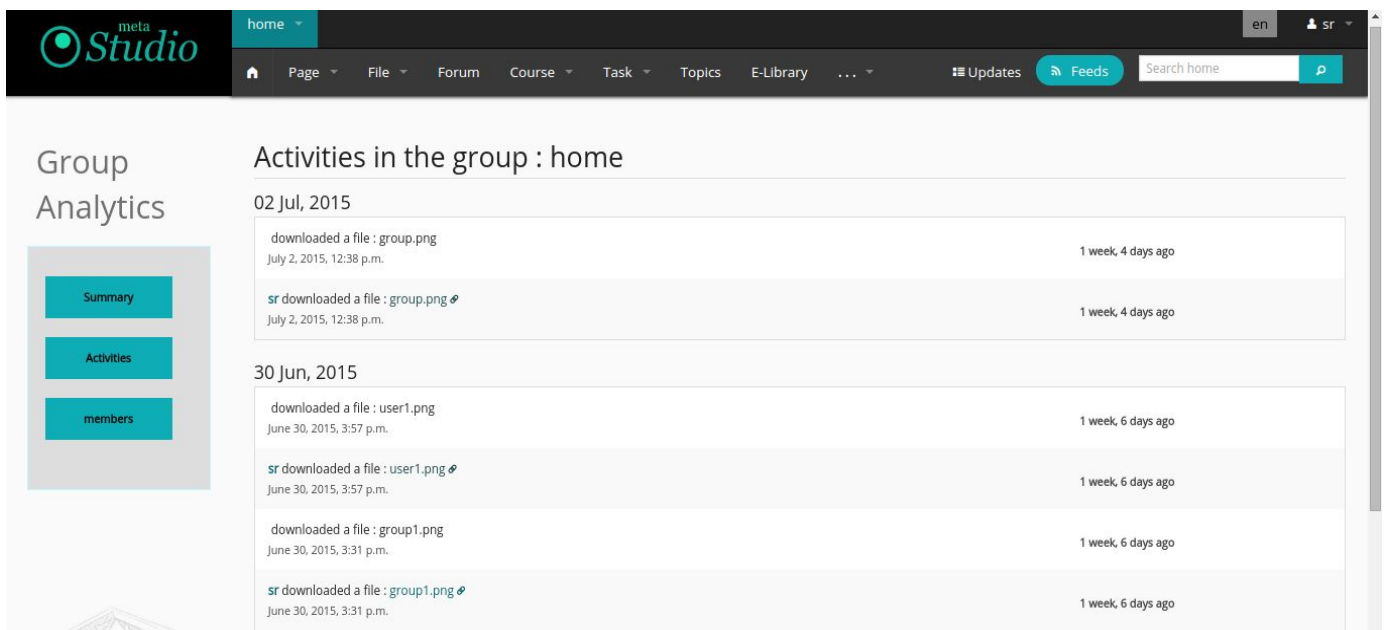


Further the view **group\_app\_activities** is called when the link symbol is accessed. It returns the list of all activities by the members of a group in that particular app.

Next image shows the recent activities in the group. This is similar to overall activities, except that the activities are of the past seven days only.



The next image shows the details of **activities**, in chronological order:



This view displays the members of the group, sorted in decreasing order of their contributions, number of contributions in each gapp, and their email id. Clicking on the member name returns a list of all the content (pages, file, forum, etc.) created by the member. "None" user represents a user who has been deleted.

The screenshot shows a web interface for 'Group Analytics'. The top navigation bar includes 'Page', 'File', 'Forum', 'Course', 'Task', 'Updates', 'Feeds', and a search bar for 'test group'. The left sidebar has 'Summary', 'Activities', and 'members' buttons. The main content area is titled 'Contributors' and displays six user cards:

Contributor	Email	Total Contributions	Content Breakdown
None		111	0 PAGES · 0 FILES · 0 FORUMS · 0 THREADS · 0 REPLIES · 0 TASKS
himanshu	bitshsdhoni1995@gmail.com	42	1 PAGES · 5 FILES · 0 FORUMS · 0 THREADS · 0 REPLIES · 1 TASKS
f2013187	f2013187@pilani.bits-pilani.ac.in	31	2 PAGES · 1 FILES · 1 FORUMS · 1 THREADS · 1 REPLIES · 0 TASKS
neal_caffrey	neal@caff.com	22	0 PAGES · 1 FILES · 1 FORUMS · 1 THREADS · 1 REPLIES · 0 TASKS
jon_snow	ylgrette_snow@wintefell.sevenKingdoms	3	1 PAGES · 0 FILES · 1 FORUMS · 0 THREADS · 0 REPLIES · 0 TASKS
rachel	rachel@gmail.com	3	0 PAGES · 1 FILES · 0 FORUMS · 0 THREADS · 0 REPLIES · 0 TASKS

## 5. RSS-FEEDS

### What is RSS ?

RSS (Rich Site Summary) is a format for delivering regularly changing web content. Many news-related sites, weblogs and other online publishers syndicate their content as an RSS Feed to whoever wants it.

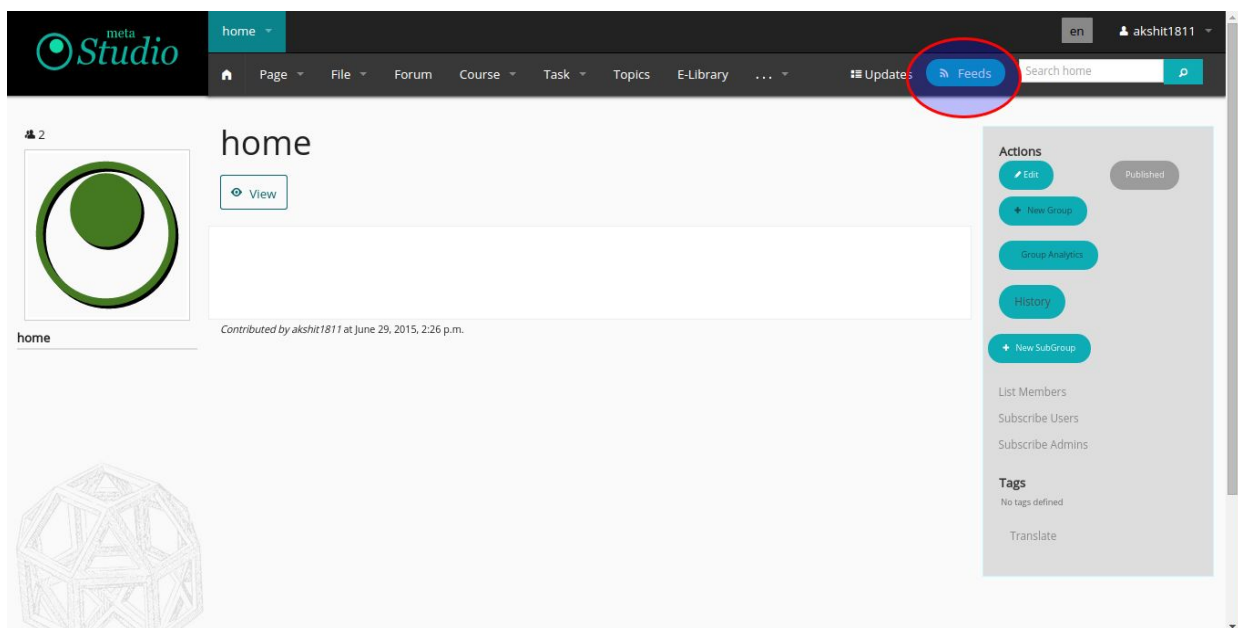
### Why RSS ?

RSS solves a problem for people who regularly use the web. It allows us to easily stay informed by retrieving the latest content from the sites we are interested in. We save time by not needing to visit the site. We ensure our privacy, by not needing to join each site's email newsletter.

### HOW IS IT DONE ?

To make use of RSS, an RSS reader, or aggregator is needed. An RSS aggregator can be a stand-alone application, or a plug-in for another program. Some web browsers, such as Firefox and Safari RSS, have RSS readers built in.

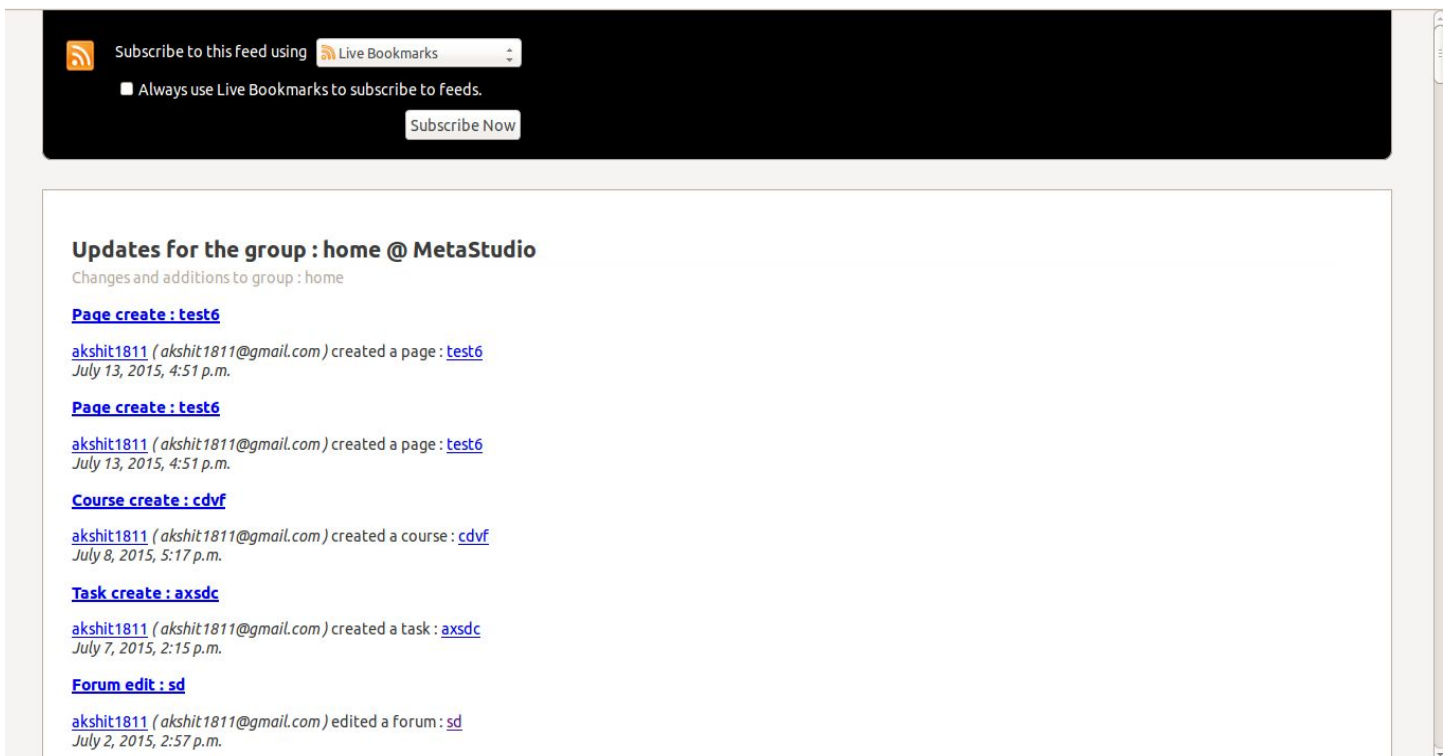
A **“Feeds”** button is placed next to the **“Updates”** at the top as highlighted in the figure below :-





The view for this is written in “**feed.py**”

It is specific to a group we are currently in. Any updates such as creation, editing , deleting of a forum or file or page, etc can be accessed by pressing this button. Also, a user can subscribe to the feeds whereby any update to the group will be mailed to him. Without visiting the site, he will be notified of any changes in the group.



Subscribe to this feed using

Always use Live Bookmarks to subscribe to feeds.

### Updates for the group : home @ MetaStudio

Changes and additions to group : home

**Page create : test6**

[akshit1811](#) ( [akshit1811@gmail.com](#) ) created a page : [test6](#)  
July 13, 2015, 4:51 p.m.

**Page create : test6**

[akshit1811](#) ( [akshit1811@gmail.com](#) ) created a page : [test6](#)  
July 13, 2015, 4:51 p.m.

**Course create : cdvf**

[akshit1811](#) ( [akshit1811@gmail.com](#) ) created a course : [cdvf](#)  
July 8, 2015, 5:17 p.m.

**Task create : axsc**

[akshit1811](#) ( [akshit1811@gmail.com](#) ) created a task : [axsc](#)  
July 7, 2015, 2:15 p.m.

**Forum edit : sd**

[akshit1811](#) ( [akshit1811@gmail.com](#) ) edited a forum : [sd](#)  
July 2, 2015, 2:57 p.m.

This is the view obtained on clicking feeds button in Firefox, or any browser with RSS reader.

If the browser does not have a RSS reader, the XML document pertaining to RSS feed is displayed.

## 6. REGISTERING CUSTOM EVENTS through AJAX

The Benchmark collects the data for all the function calls through specific URLs as well as the functions invoked inside the VIEW functions. Hence, our model for analytics till now, is only based upon the data that we get from the method calls that is being registered by the Benchmark model. Hence, the above flow does not facilitates for the Events that occur on the front-end and is not 'communicated' to the back-end.

To provide for these events to be registered in the database, a separate handle is created to directly register the event from the front-end to the `analytics_collection` model by means of AJAX calls to a specific URL.

### 6.1 HOW TO ?

Registering Custom events is being included as the part of Analytics module for MetaStudio. The events can be registered by an AJAX request to the URL :

`http://<domain-name>/<group_id>/analytics/custom_events`

The specifics of the event registration has been wrapped into a JavaScript function `registerEvent(obj, properties, action)`. This function is included in the static JavaScript file `analytics.js` which gets included into every template file that extends `base.html` by means of a small code snippet that has been included in the `base.html` file.

Hence, registering for custom events is binding the `registerEvent()` function to the specific event or trigger on the website with the specific details about the event object and action that define the event.

### 6.2 registerEvent(obj, properties, action)

The function works to insert the event data into the `analytics_collection` collection. It takes in the following parameter about the event type and object :

**obj** (*string*) : defines the name of the app or the specific activity like image, video, file, page, forum etc. This appears in detailed activities view as : Neo created a **file**, where obj is 'file'.

**properties** (*dict*) : defines the dictionary of properties of the **obj**. This has a few compulsory properties like **name** (defines the name of the obj), **url** (same as the url attribute of the node), **id** (node ID of the obj). Other properties can include mime-type, sub-app objects, connected nodes etc.

**action** (*dict*) : defines the type of action. It is a dictionary that takes in two attributes, **'key'** : one word to define the event action like add, create, delete etc. and **'phrase'** : the verb phrase that would appear in the semantic representation of the activity.

## 6.3 USAGE

Take the example of registering the activity to view a specific video on metaStudio platform. We want to bind **registerEvent()** to the event of a button click.

HTML :

```
<button id="view-video" class="">PLAY VIDEO</button>
```

JavaScript ( using jQuery ) :

```
$("#view-video").click(function() {  
    var obj = "video"  
    var action = {  
        'key' : 'view',  
        'phrase' : 'viewed a'  
    };  
    var properties = {  
        'id' : "j3452k45hg234kj5h34j5h34",  
        'link' : 'video',  
        'name' : 'Learning German Part 1'  
    };  
    registerEvent(obj, properties, action);  
});
```

## 7. TECHNOLOGIES USED

The project generally lends from popular the elements of development used for Web. MetaStudio is deployed using python-based Django Web Framework and uses MongoDB for storage on the concept of RDF (Resource Description Framework).

### 1. JAVASCRIPT

JavaScript (JS) is the web scripting language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter DOM that is displayed.

### 2. HTML 5

HTML5 is a core technology markup language of the Internet used for structuring and presenting content for the World Wide Web. Its core aims have been to improve the language with support for the latest multimedia while keeping it easily readable by humans and consistently understood by computers and devices (web browsers, parsers, etc.).

### 3. CSS 3

Cascading Style Sheets (CSS) is a style sheet language used for describing the look and formatting of a document written in a markup language. Along with HTML and JavaScript, CSS is a cornerstone technology used by most websites to create visually engaging web pages, user interfaces for web applications, and user interfaces for many mobile applications.

### 4. MONGODB

MongoDB (from "humongous") is a cross-platform document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON- like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster.

### 5. DJANGO

Django is a free and open source web application framework, written in Python, which follows the model–view–controller architectural pattern. It is maintained by the Django Software Foundation (DSF), an independent organization established as a non-profit. Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings, files, and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

## **6. MONGOKIT, Django-mongokit**

MongoKit is a python module that brings a structured schema and validation layer on top of the great pymongo driver. It has been written to be as simple and light as possible with the KISS and DRY principles in mind. MongoKit uses plain python types to describe document structure. It is fast and brings many features like document auto-reference, custom types or i18n support.

## **7. FOUNDATION 5**

Foundation 5 CSS framework is a free collection of tools for creating websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. Since version 2.0 it also supports responsive design.

## **8. JQUERY**

jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. Used by over 80% of the 10,000 most visited websites, jQuery is the most popular JavaScript library in use today. jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities developers to create plug-ins on top of the JavaScript library.

## **9. AJAX**

Acronym for Asynchronous JavaScript and XML is a group of interrelated Web development techniques used on the client-side to create asynchronous Web applications. With Ajax, Web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page. Data can be retrieved using the XMLHttpRequest object.

## **10. GIT**

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It is used for version tracking of a project having multiple feature and parts, collaboration, error resolving. MetaStudio is hosted over GitHub which implements git to maintain a code project.

### **11. JSON and XML**

These are the standards for data exchange over the web. Both consists of key, value pairs and can be nested. Thus, the data has a tree structure that makes readability and parsing easy in communication and interpretation

### **12. REGEX**

Regular expression is a sequence of characters that define a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations. Regular expressions are so useful in computing that the various systems to specify regular expressions have evolved to provide both a basic and extended standard for the grammar and syntax; modern regular expressions heavily augment the standard.

### **13. DIFFLIB**

difflib is a python module that contains tools for computing and working with differences between sequences. It is especially useful for comparing text, and includes functions that produce reports using several common difference formats.

## 8. FUTURE SCOPE

Analytics for any web application defines an endless field of opportunities. Our work to implement this on MetaStudio are the initial steps towards the ultimate goal to measure activities and produce User Reports for users on this platform.

The project can be exhaustively divided into 3 separate domains and a lot can be done to improve each of them :

### 1. DATA COLLECTION

At present, we collect the data solely by analyzing the function calls through specific URLs and invoked functions. Also, we created an AJAX handle to register custom user events that will be directly analysed and stored in the **analytics\_collections**.

1. Data from the RCS (revision control system) can be used for analysis and keeping track of version upgrades.
2. Registering Custom user activities through AJAX : Writing events and triggers for registering custom data on the page.
3. **analyTHROUGHtics\_collection** data model currently uses concept of RDF to store the user activity. This model can further be extended to link the sequential activities to study the Users' control flow on the website.

### 2. SEGREGATION and ANALYSIS

New methods for data analysis can be deployed to draw conclusions from the existing data in **Benchmarks** and **analytics\_collection**.

1. **Generating User Reports** for members in the Group and exporting in the form of PDF.
2. **Real-time updation** and rendering of events taking place on the platform. Our methodology depends on passive updation of the analytics data which is then rendered to the user. Scope for real-time updation is left untouched but not impossible to handle. If we are able to clear out data redundancy from Benchmark collection at real-time, everything becomes cakewalk !

3. The user should have an option to **modulate the tracking** by Metastudio. This plays an important role in concern to the privacy and should be implemented before major advancements.
4. **Course Analytics** : Course specific activities and analysis should be beneficial addition to the module. However, this can be implemented only after the Course module is fully functional and Live. Course Analytics further opens a vast opportunity for data analysis and representation.
5. A major requirement of this project is to analyze User **Click Streams** so as to study the user's whereabouts on the website. This data can be used to optimize the platform according to the needs and requirements of the user.

### 3. DATA REPRESENTATION

Visualization and Graphical representation of user activities to study usage patterns and behavioral aspects. Certain graphs that may be valuable :

1. **Radial Graph** : To study the % activities for each gApp like Page, Files, Forums, Images, Videos, Tasks, Events, Courses etc.
2. **Calendar View** : To show the variation in user activities over the days and months.
3. **Bubble Graph** : To show the amount of contributions by each user in a particular group.
4. **Timelines** : To represent complete user activities or specific activities over range of time in a timeline format.

The Analytics model is currently deployed as a gApp, being integrated into the metaStudio platform. However, to achieve greater computation and analysis, the project could well be established as a separate web-service so as to make it completely unattached with the platform. This will greatly enhance the working and managing capabilities of the data and will provide better infrastructure for complex computation and analysis.



## IX. CONCLUSION

The aim of the project was to implement Analytics on the MetaStudio platform. This include tracking the user activities on the website in various gApps like page, forum, files, tasks, events, course etc. , processing the 'relevant' data and storing in the database. This collection, then consist of useful set of user activities that can be used to generate user reports and draw useful conclusions.

The project was the initial step to bring analytics to metaStudio. The following were the major steps that were implemented over the course of this project :

- Changing the **Benchmarks** model to accommodate more details about the called URL like the **username, session\_key, action, has\_data** (if the request has POST/GET data).
- Processing the raw data from **Benchmarks** model for filtering unwanted actions, removing redundancy, and then, analyzing the 'useful user activities'.
- The analyzed Activity is being stored in '**analytics\_collection**' collection. This data is organized using the RDF schema wherein each activity has 3 components : subject, predicate, object. Subject denotes who did the action (user). Predicate denotes the action of the subject, while object refers to the entity on which the action is being performed.
- The updation of the analytics database is done using the batch processing mechanism, wherein the partial data which is not in the **analytics\_collection** is processed and the collection is updated, each time the user accesses the Analytics module.

The analytics model basically provides for User and Group at present.

- The **User Analytics** have 2 views : Summary (statistical data for user's activity), List Activities (detailed activities of the user). Also, the user can view has app-specific activities which currently includes page, file, forum and task.

- Likewise, **Group Analytics** provides activities for a particular Group. It is visible only to the group admins and the super users of the platform. It has : Summary (statistical data for activities in the group), List Activities (detailed activities in the group), Members (member contributions in the group), Member Info (Individual Activity of individual member in the group).

Group **Update Feeds** implements web syndication for the updates in a particular group. The user can subscribe to these RSS feeds to get info about the specific activities of the group like creation, editing and deletion of group resources. It has been implemented inside Group Analytics, but as a separate module in the MetaStudio. These updates can be subscribed to by all the users currently part of a particular Group.

The user interface for the Analytics and Feeds module has been designed keeping in mind the needs and convenience of the user. The framework uses AJAX for asynchronous data communication and updation from the server, implementing switching between different Views of Analytics. The templates produce are simple and intuitive in design, borrowing UI Features from the existing metaStudio template.

The development phase shall be followed by testing phase where the efficiency, result and workflow of the entire Analytics app shall be tested in a multi-user environment, to test how the logic scales as more and more data is added to the metastudio framework. This will further enable us to get a feedback of the algorithm and test the robustness of the Analytics module.

## X. REFERENCES

- Bengtsson, P. How and why to use django-mongokit (a.k.a Django to MongoDB).  
<http://www.peterbe.com/plog/how-and-why-to-use-django-mongokit>
- Berryman, J. Semantic Search With Solr And Python Numpy.  
<http://www.opensourceconnections.com/blog/2013/08/25/semantic-search-with-solr-and-python-numpy/>
- Caraciolo, M. Collaborative Filtering : Implementation with Python!  
<http://aimotion.blogspot.in/2009/11/collaborative-filtering-implementation.html>
- django. Django Documentation. Retrieved from  
<https://docs.djangoproject.com/en/1.6/intro/>
- mongoDB. MongoDB Manual  
<http://docs.mongodb.org/manual/>
- tutorialspoint.com. (2014, May 20). Python Programming.  
<http://www.tutorialspoint.com/python/>
- GitHub. The Official Github Documentation and Guides  
<https://help.github.com/>, <https://guides.github.com/>
- Git SCM. The Official Git Documentation  
<https://git-scm.com/documentation>
- Atlassian Git. Getting Git Right, GIT Manual  
<https://www.atlassian.com/git/>
- Foundation Official Documentation

<http://foundation.zurb.com/docs/>