Graduation Project Plan

Fontys University of Applied Sciences

"Open the Chronicle"

Author	:	Eko Adi Prasetyo Gunawan
Student Number	:	2154590
School Tutor	:	Erik H.J.D. van der Schriek
Company Tutor	:	Laurent Eschenauer
Version	:	0.5
Date	:	05/29/10

Version Control

VERSION	DATE	DESCRIPTION	REASON
0.1	Feb 17, 2010	Initial version	
0.2	Feb 24, 2010	Minor changes in chapter 1,2 and 4.	Revision based on the feedbacks from tutors.
0.3	Mar 08, 2010	The end date of Design Phase is changed into 19th of March.	The Design Phase period need to be extended.
0.4	Mar 26, 2010	'Duration' column in project schedule changed into 'Effort'.	To show how long (effort in day) each task is done.
0.5	May 29, 2010	1. Report's appearance changed. 2. Version control added.	1. To follow the appearance of Final Report. 2. To track the changes in each version.

Table of Contents

1. Introduction

Vodafone Group Plc is the world's leading mobile telecommunication company. The Group's mobile subsidiaries operate under the brand name 'Vodafone'. Vodafone provides services to make their customer stay connected to the people and all important things for them. Group R&D within the company is an international team which applies research in mobile and internet communication and their related applications. **Betavine**[1] is a testing ground for the latest concepts and technologies in mobile and internet applications run by the R&D team.

1.1 Current Situation

Vodafone R&D will soon launch a service called **Betavine Chronicle**[2] which based on open source lifestreaming platform **Storytlr**[3] and hosted on Betavine. Chronicle offers lifestream widgets, story widgets and also different pages with blog functionality. It is also integrates third party services such as Twitter, Youtube, Picasa and many more. Using Chronicle is like integrating **web 2.0** into one website.

Surya Wijaya Madjid as a student trainee at Vodafone has a project to develop an android based mobile application which can use the services provided by Betavine Chronicle. However, currently there is no possibility for third party developers to use the Chronicle service. There is still no API available for the Chronicle service.

1.2 Project Description

Building a **REST**[4] **API** for third party developers especially for the other trainee which enable them to use the service provided by Chronicle such as view stories and lifestream activities from Chronicle, also change status, post blog contents, share links, etc. to Chronicle from their application.

Along with building the **REST API**, an API documentation should also be released in order to give the developers the knowledge of the services provided from the API also to help them to use the appropriate services.

The API provided will run on the live production server, therefore both the code and the documentation have to meet the production quality. That means the code must follow the **Zend Framework Coding Standard for PHP**[5] and the API documentation must be easy to understand by the third party developers. It might be released as open source code, on top of the **Betavine** platform. Appendix C shows the overview schema of the project.

1.3 Project Justification

As mentioned above, Chronicle offers integration with several third party services, in other words, using the Chronicle services is also using some services from another web

[1] http://www.vodafone.com/start/innovation/betavine.html
[2] http://chronicle.betavine.net
[3] http://storytlr.googlecode.com
[4] http://en.wikipedia.org/wiki/Representational_State_Transfer
[5] http://framework.zend.com/manual/en/coding-standard.html

application such as Twitter, Youtube, Picasa, etc..

For developers it will be easier to learn and use one interface which can be used for several application than learn several different interfaces for each application. The API documentation provided along with the API will help the developers to understand how to use the available services of Betavine Chronicle.

It is also easier for users to use only one application for accessing several applications. For example, users can change their Twitter status from Chronicle.

1.4 Formal Client

Company : Vodafone Group R&D Lab – NL

Address : Avenue Ceramique 300, 6221KX Maastricht

Representative : Laurent Eschenauer

Tel : +31 (0) 61 195 4411

E-mail : laurent.eschenauer@vodafone.com

Website : www.vodafone-rnd.com

1.5 Project Leader

Name : Laurent Eschenauer

Company : Vodafone Group R&D Lab – NL

Tel : +31 (0) 61 195 4411

E-mail : laurent.eschenauer@vodafone.com

2. Management / Technical

2.1 Management Objectives

The project objectives, assumptions, and constraints of managerial and technical part will be described in the following section.

2.1.1 General Objectives

- The project phasing included in this document will help the author to do each activity in the development process of the project in time.
- Project risks are to be assessed and documented. If necessary, and upon occurrence of risk, action should be taken according to the Risk Management Plan.
- The client and school tutor should be made immediately aware of any possible delays to project deliverables. The relaying of such information to all clients must be documented.
- All remarks and feedbacks of the discussion during the meeting will be documented in form of action points to help the author have a clear vision of the meeting.
- **Git**[6] tool is used to maintain and update the resource, and also accessible by the client.

2.1.2 Assumption

Several assumptions were made in order to come up with the best result on the product. The author will try to look on them during the project realization.

- Resource Assumptions
    - The company will provide a computer/laptop to be used by the author during development.
    - The company tutor and school tutor will provide assistance and guidance where needed.
    - The client, company tutor, and school tutor will provide feedback on the product(s) released and any documents/reports during development process.

- Delivery Assumptions
    - The complete working REST API should be available in the Transfer Phase on the last release.

[6] http://git-scm.com

  - A complete API documentation should be included along with the working API.
  - All source code and its developments documentations will be handled over to the client.

- Environmental Assumptions
  - No promises will be made that can affect the project.
  - Issues will be resolved in a timely manner.
  - The project organization as described in this project plan will be implemented.
- Functionality Assumptions
  - The scope of the project is limited to what is described in this project plan.

2.1.3 Management Constraints

- The project team only consists of those who are mentioned on the organizational chart in this project plan.
- The project must fill the minimum system requirements as agreed by both the client and tutor.
- The less important features of the project may be dropped / not implemented if the implementation time is considered too less.

2.1.4 Technical Constraints

- The REST API provided will run in live production server together with Betavine Chronicle.
- To be able to use the service provided by the API, the third party application should be able to connect to the server where the API is run.

2.2 Project Controls

The Gantt Chart included in this document will be used to monitor the project progress. It will be updated in case of changes in project phasing and schedule.

The author and the company tutor will hold a weekly meeting. The author will report the status of his work and the company tutor will give feedback and assistance/guidance if needed. There will be 30 minutes provided by the company tutor everyday to answer questions.

2.3 Risk Management

The section below is produced to identify possible risks related to the project. It also

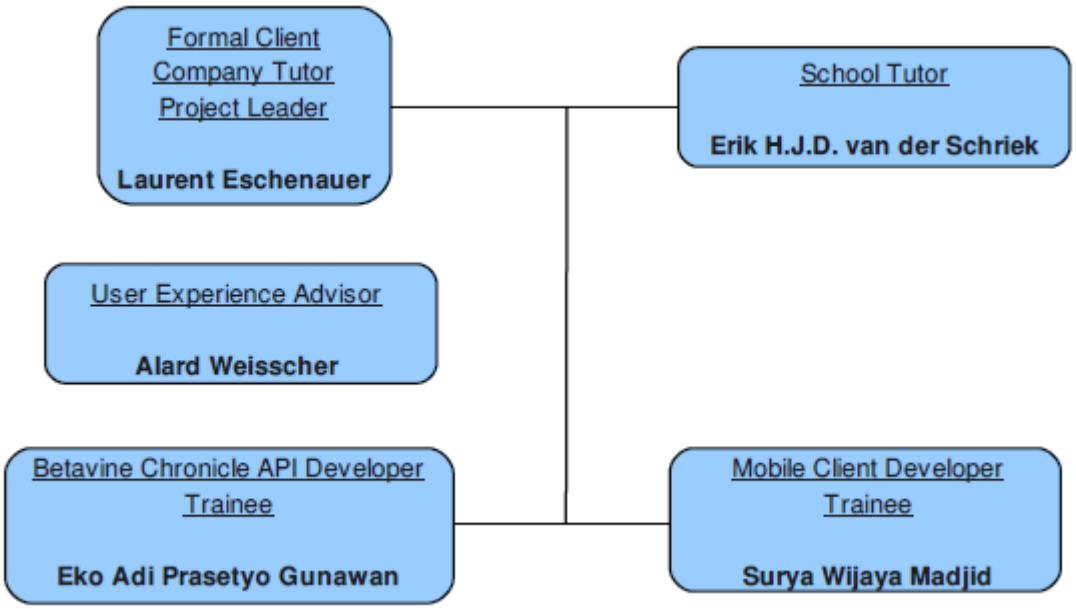
includes a course of action to be taken in the event of a problem. Additional risks which are found during the development period will also be added.

- Lack of knowledge required for the project **(High Probability)**. There are a lot new things need to be studied by the author in order to build the REST API of Betavine Chronicle, therefore research regarding on it could take a lot of time and might delay the project progress **(High Impact)**. **Mitigation:** The research will be conducted throughout the development process, combined with consultation with the company and school tutor.
- The other trainee doesn't finish the deliverables at the time specified in the project phasing **(High Probability)**. The author needs the application provided by the other trainee to test whether the API working correctly or not. If the other trainee cannot provide the application in time, the project may be delayed **(Medium Impact)**. **Mitigation:** Work together as a team, give feedback and monitor the progress of each other. Build a **unit testing**[7] which enables the author to test the API without using the other trainee's application.
- Bugs from the Betavine Chronicle service **(Low Probability)**. There might by some bugs appear which can delay the project **(Medium Impact)**. **Mitigation:** In case of bugs appearing (participate in finding the bugs), report it immediately and continue the project development with assumption that the service works well and then test it after the bug(s) has been fixed.
- System crash or data lost during the project development **(Low Probability)**. Something bad could happen during the development process, such as broken laptop or system crash because some changes to the code. However, the use of git tool can significantly decrease this risk **(Low Impact)**. **Mitigation:** Regularly back up the data into external source using git tool.

2.4 Project Staffing – Organizational chart

The following organizational chart displays the hierarchical structure of the whole project group (this graduation project is part of a bigger project).

[7] http://en.wikipedia.org/wiki/Unit_testing



2.5 Communication Plan

The list of people involved in this project is described below including the role, contact, and activities of each person.

Name	Contact	Activities	When
Eko Adi Prasetyo Gunawan	e.gunawan@student.fontys.nl +31 (0) 61 952 1718 Leenderweg 218, 5644AB Eindhoven	Finish the project in time with the best result by asking for assistance, guidance, and feedback from other stakeholder.	Discussion and testing regarding the project can be done at anytime when needed.
Surya Wijaya Madjid	s.madjid@student.fontys.nl +31 (0) 61 624 8188 Leenderweg 218, 5644AB Eindhoven	Provide a mobile client application which is using the Betavine Chronicle services. Work together with Eko in order to test the application and monitor the progress each other.	Discussion and testing regarding the project can be done at anytime when needed.
Laurent Eschenauer	laurent.eschenauer@vodafone.com +31 (0) 61 195 4411 Avenue Ceramique 300, 6221KX Maastricht (Vodafone Group R&D Lab)	Act as a client, give the requirements specifications. Give assistance, guidance, and feedback to the student trainee as a company tutor.	30 minutes available everyday around 12.00 for asking questions and weekly meeting every Thursday for 1 hour.
Erik H.J.D. van der Schriek	e.vanderschriek@fontys.nl +31 (0) 65 377 5694 Rachelsmolen 1, 5612MA Eindhoven (Fontys Building R1 Room 4.48)	As a school tutor, give feedback of any technical document produced by the student trainee. Monitor the progress of the student trainee.	There will be 2 visits to the company (will be determined later on) and the author can ask for guidance when needed.
Alard Weisscher	alard.weisscher@vodafone.com +31 (0) 62 129 6199 Avenue Ceramique 300, 6221KX Maastricht (Vodafone Group R&D Lab)	Give advise about the user interface of the application (especially to Surya)	Any questions regarding user interface can be asked anytime (working time).

2.6 Technical Processes

This section is describing the software development methodology used for this project and also list of software used during the development.

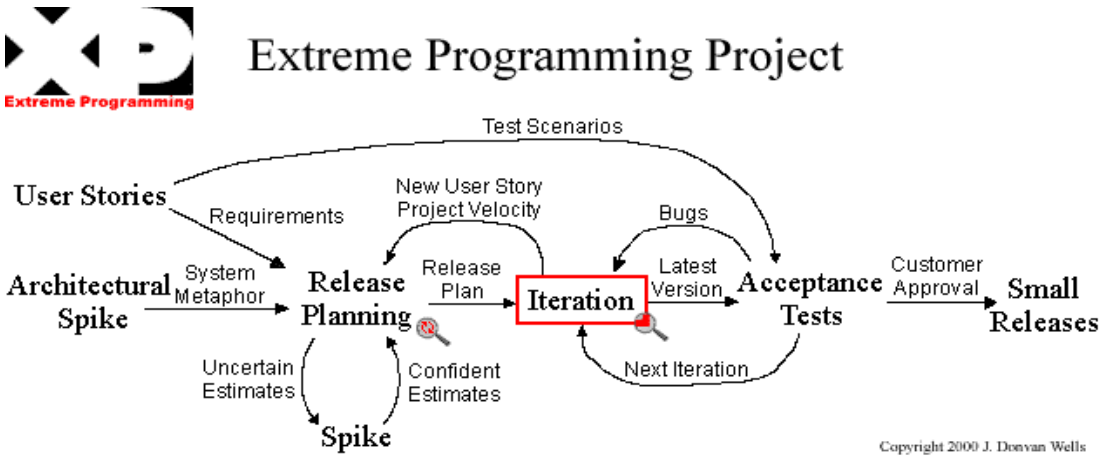
2.6.1 Project Methodology

There will be two methods used to develop this project. Waterfall model will be used as the main methodology to make the planning and scheduling easier. However, both the author and the client are agreed to adapt the **Extreme Programming**[8] as a method to be implemented during the implementation phase.

Extreme programming improves the project in five essential ways; communication, simplicity, feedback, respect, and courage. Feedback is obtained by testing the software starting on early period, the system is delivered to the client as early as possible and changes are implemented as suggested. In this methodology, the client will take part in the development team; therefore the communication between the developer and the client can be done very easily.

Client can change the requirements during the development process, the requirement specified in the functional requirements document is the initial requirements which can be added or reduced. To make sure the API will work properly and delivered in time, there will be 3 releases where each release will focus on implementing several requirements depends on the requirement's priority. Each release will be divided into iterations, therefore acceptance test and bug fixing will be conducted before moving into the next iteration. If there isn't enough time, some requirements will not be implemented; otherwise more requirements could be added. Pair Programming will not be implemented because there is not enough project members in the team.

Below is the flow chart of Extreme Programming methodology.



2.6.2 Project Software

The following is a list of software to be used during development:

- OpenOffice.org Word Processors.
- OpenOffice.org Spreadsheet.
- Eclipse Galileo (PHP).

[8] http://www.extremeprogramming.org

- Apache2.
- MySQL.
- Git tools.

3. Project Statement

3.1 Project Scope

This project is about building a REST API of Betavine Chronicle service which enables the third party developers to read and write data to Chronicle (more details about it will be mentioned in the Functional Requirements document).

The requirements list will be classified in a **MOSCOW** table, the requirements which are classified as "Must Have" must be implemented, some additional requirements will be implemented if there is enough time before the period of performance of this project is finished.

3.2 Location of Work

The whole development process will be conducted at Vodafone office which is located on Avenue Ceramique 300, 6221KX Maastricht. This location will also be used for meetings with client and school tutor.

3.3 Period of Performance

Start Date : 1 February 2010

End Date : 30 June 2010

3.4 Key Deliverables

- The working REST API of Betavine Chronicle service.
- API documentation which explains how to use the services provided by the API.
- All document related to the development process during the project: Project plan, functional requirements, and technical manual.

3.5 Acceptance Criteria

The acceptance of the project is to be based upon the minimum requirements specified in the Functional Requirements document. The project will be deemed as succeed only when the client has tested and signed the project.

4. Project Phasing

4.1 Initiation Phase

Activities:

- Get acquainted to the company.
- Preparing questions for the client.
- Conduct research about all the required things to develop the project.
- Understanding the concept of building the REST API.
- Writing the project plan.

Deliverables:

- Project Plan.

Duration:

- 1 February 2010 – 19 February 2010

4.2 Design Phase

Activities:

- Defining the functional and non-functional requirements based on the client's information.
- Making the draft version of API documentation.
- Making high level architecture diagram.
- Conducting feasibility study.

Deliverables:

- Functional Requirements document – including the priorities.
- Draft version of API documentation.
- Feasibility study report – report of experiments on the technologies which will be used to build the project to make sure that the project is feasible.

Duration:

- 22 February 2010 – 19 March 2010

4.3 Implementation Phase

Activities:

- Building the API.
- Testing each functionalities in the API.
- Fixing bugs and problems.
- Making the code documentation.
- Making the API documentation.
- During this phase, there will be 3 releases with improved functionality and quality of the product in each release.

Deliverables:

- (see deliverables of the Transfer Phase below)

Duration:

- 15 March 2010 – 30 June 2010

There will be weekly iterations during this phase, which implement specific functionalities in each iterations.

4.4 Transfer Phase

Activities:

- Conduct acceptance test with the client.
- Review of writing code and documentation.
- Deliver final version of the product and all of its documentations to the client.

Deliverables:

- Working API for each release.
- API documentation for each release.
- Code documentation.
- Source code of the API.
- Final project report (contains all documentations required, delivered at the 3rd release).

Duration:

- 1st Release : 19 April 2010 – 23 April 2010
- 2nd Release : 31 May 2010 – 4 June 2010

- 3rd Release : 28 June 2010 – 30 June 2010

4.5 Graduation Finalization Phase

Activities:

- Making the final graduation report and presentation.
- Preparing the graduation presentation.
- Conducting presentation and defence.

Deliverables:

- Final graduation report.
- Graduation presentation slide.

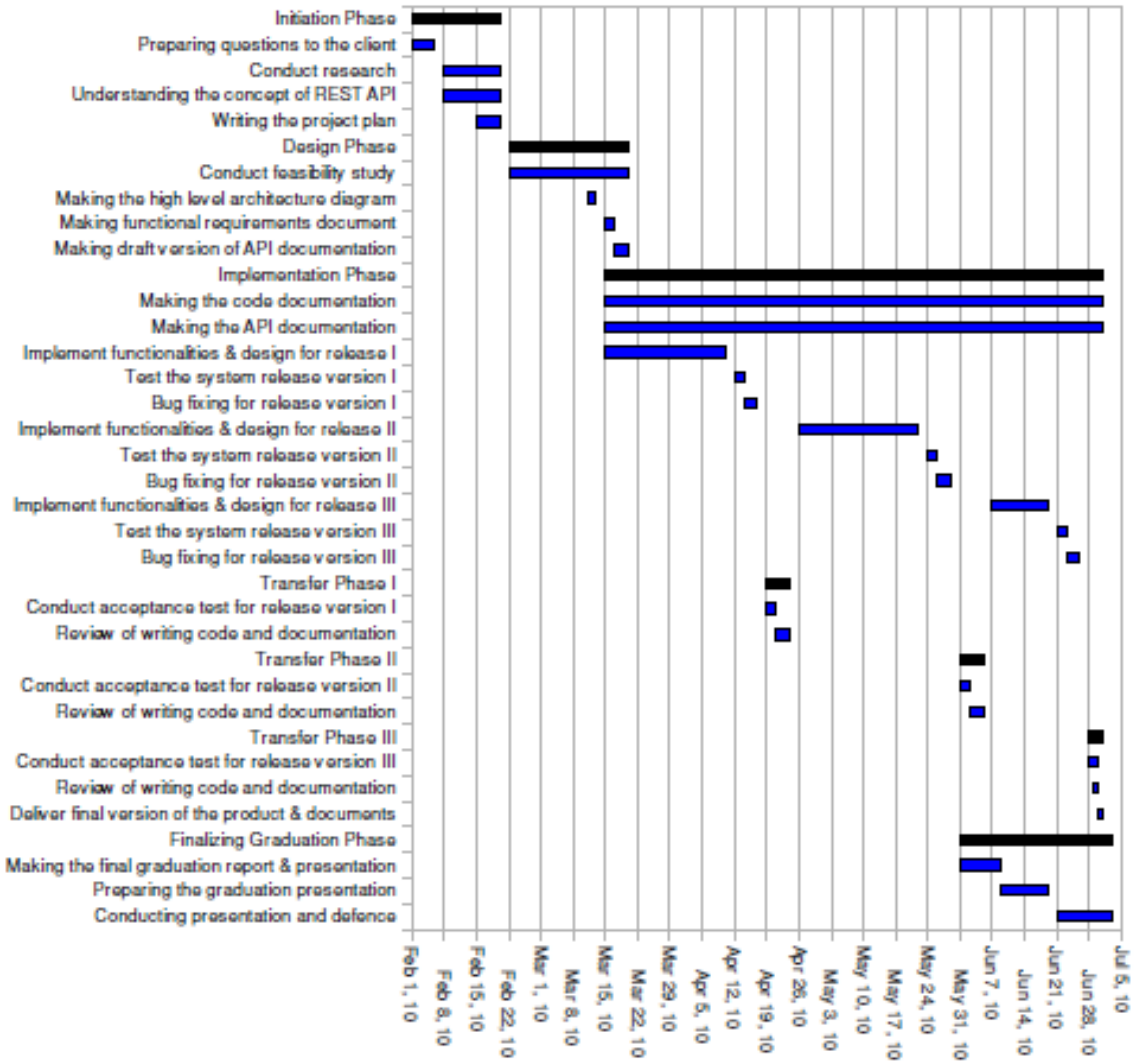
Duration:

- 31 May 2010 – 2 July 2010

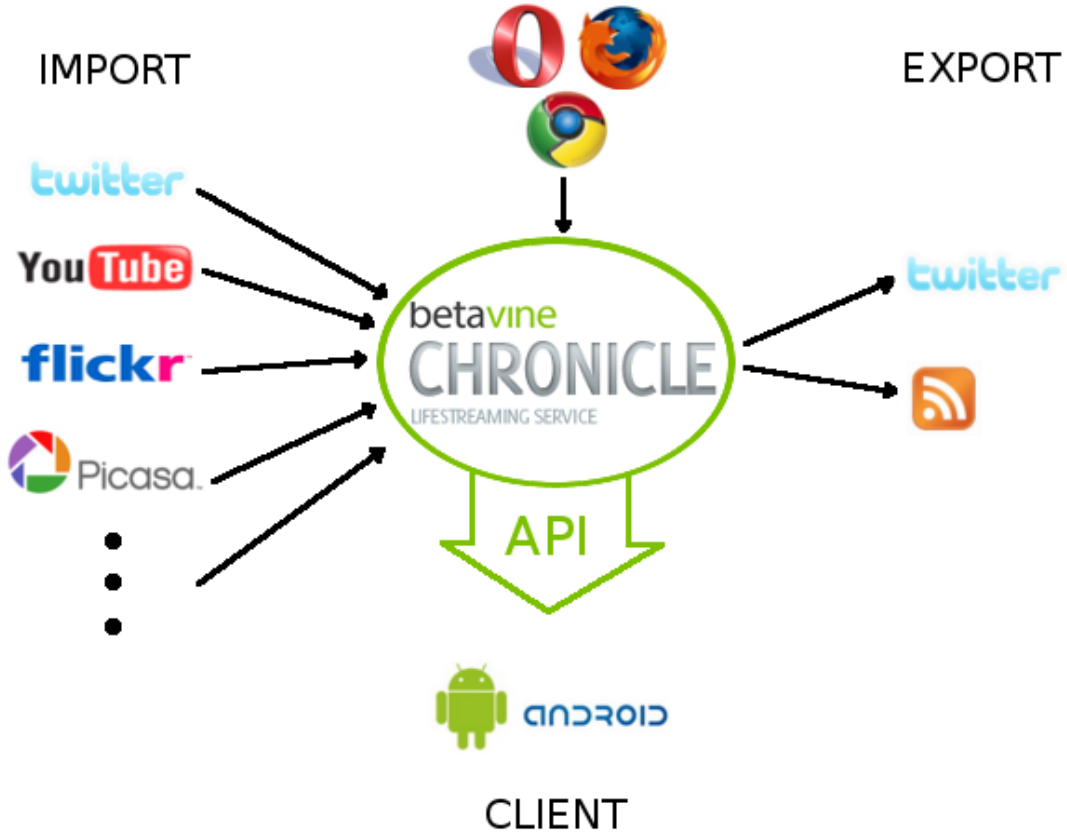
Appendix A – Schedule

	Task Name	Start	Finish	Effort
1	**Initiation Phase**	**01 Feb 2010**	**19 Feb 2010**	**15 days**
2	Preparing questions to the client	01 Feb 2010	05 Feb 2010	2 days
3	Conduct research	08 Feb 2010	19 Feb 2010	5 days
4	Understanding the concept of REST API	08 Feb 2010	19 Feb 2010	3 days
5	Writing the project plan	15 Feb 2010	19 Feb 2010	5 days
7	**Design Phase**	**22 Feb 2010**	**19 Mar 2010**	**20 days**
8	Conduct feasibility study	22 Feb 2010	19 Mar 2010	14 days
9	Making the high level architecture diagram	11 Mar 2010	12 Mar 2010	2 days
10	Making functional requirements document	15 Mar 2010	16 Mar 2010	2 days
11	Making draft version of API documentation	17 Mar 2010	19 Mar 2010	2 days
12	**Implementation Phase**	**15 Mar 2010**	**30 Jun 2010**	**53 days**
13	Making the code documentation	15 Mar 2010	30 Jun 2010	5 days
14	Making the API documentation	15 Mar 2010	30 Jun 2010	5 days
15	Implement functionalities & design for release I	15 Mar 2010	09 Apr 2010	13 days
16	Test the system release version I	12 Apr 2010	13 Apr 2010	1 days
17	Bug fixing for release version I	14 Apr 2010	16 Apr 2010	2 days
18	Implement functionalities & design for release II	26 Apr 2010	21 May 2010	13 days
19	Test the system release version II	24 May 2010	25 May 2010	1 days
20	Bug fixing for release version II	26 May 2010	28 May 2010	2 days
21	Implement functionalities & design for release III	07 Jun 2010	18 Jun 2010	8 days
22	Test the system release version III	21 Jun 2010	22 Jun 2010	1 days
23	Bug fixing for release version III	23 Jun 2010	25 Jun 2010	2 days
24	**Transfer Phase I**	**19 Apr 2010**	**23 Apr 2010**	**5 days**
25	Conduct acceptance test for release version I	19 Apr 2010	20 Apr 2010	2 days
26	Review of writing code and documentation	21 Apr 2010	23 Apr 2010	3 days
27	**Transfer Phase II**	**31 May 2010**	**04 Jun 2010**	**5 days**
28	Conduct acceptance test for release version II	31 May 2010	01 Jun 2010	2 days
29	Review of writing code and documentation	02 Jun 2010	04 Jun 2010	3 days
30	**Transfer Phase III**	**28 Jun 2010**	**30 Jun 2010**	**5 days**
31	Conduct acceptance test for release version III	28 Jun 2010	29 Jun 2010	3 days
32	Review of writing code and documentation	29 Jun 2010	29 Jun 2010	1 days
33	Deliver final version of the product & documents	30 Jun 2010	30 Jun 2010	1 days
34	**Finalizing Graduation Phase**	**31 May 2010**	**02 Jul 2010**	**10 days**
35	Making the final graduation report & presentation	31 May 2010	08 Jun 2010	7 days
36	Preparing the graduation presentation	09 Jun 2010	18 Jun 2010	2 days
37	Conducting presentation and defence	21 Jun 2010	02 Jul 2010	1 days

Appendix B – Gantt Chart



Appendix C – Project Overview Schema



Appendix D – Approval Signature

Wednesday, February 24, 2010

Company Tutor	School Tutor
Laurent Eschenauer	**Erik H.J.D. van der Schriek**

Feasibility Study Report
Fontys University of Applied Sciences

"Open the Chronicle"

Author	:	Eko Adi Prasetyo Gunawan
Student Number	:	2154590
School Tutor	:	Erik H.J.D. van der Schriek
Company Tutor	:	Laurent Eschenauer
Version	:	0.4
Date	:	Jun 05, 2010

Version Control

VERSION	DATE	DESCRIPTION	REASON
0.1	Mar 25, 2010	Initial version	
0.2	May 14, 2010	1. Merge chapter 2 and 3. Separate each research topic into sub-chapter. 2. Minor changes in chapter 3 and 4 (previously chapter 4 and 5)	1. To more specificly explain each research. 2. To add more conclusion on proof of concept and research.
0.3	May 29, 2010	1. Added more figures and description in chapter 3 (Proof of Concept). 2. Report's appearance changed. 3. Version control added.	1. To show and explain the PoC's code. 2. To follow the appearance of Final Report. 3. To track the changes in each version.
0.4	Jun 05, 2010	Change the Project Prototype into Proof of Concept	It turned out that the application which was built during this feasibility study is not a project prototype.

Table of Contents

1. Introduction

The “API for Betavine Chronicle” (ABC) project's main goal is to create a REST API which enable the third party application to do some functionality in Betavine Chronicle. The detailed description of the project can be seen in the Project Plan.

There are a lot of terms, technologies, and standards will be used for the project, however most of those things are new for the author. Therefore, the author need to conduct a feasibility study to learn those things, to have a better overview of the project, to have a clear vision of how to finish the project, and to make sure the project can be done in time. This report is the result of the feasibility study.

This report consists of two main chapters: Research and Proof of Concept. Chapter 2, Research, explains the research conducted during the feasibility study, the questions, the criteria, the analysis, and the conclusion of each question. Chapter 3, Proof of Concept (PoC), gives information about building the proof of concept. The research's result is used to build the PoC.

2. Research

There were some decisions had to be made before building the proof of concept, therefore some small research was conducted. The following sub chapter will explain each research. The research was done by comparing the possible solutions to the criteria of ideal solution. Comparison was done by finding any information about the solutions related to the given criteria.

2.1 Which REST Implementation Should be Applied?

Zend Framework is the web application framework used to build the Betavine Chronicle service, therefore the REST API have to be built within Zend Framework environment as well.

To build the REST API within Zend Framework, either create a new REST handler or use the existing REST feature which are provided by Zend Framework is possible.

2.1.1 Criteria

Criteria of REST implementation for the project is:

- Ease of use
- Robustness

2.1.2 Analysis

There two ways of implementing REST in Zend Framework, create a new REST handler and use the existing REST support in Zend Framework.

Create a New REST Handler

The new REST handler must be able to process the client's HTTP request with the proper action. Every request have to be filtered and routed into the appropriate function so that the system can give the correct response.

There are 4 HTTP request methods used for RESTful application, that is GET, POST, PUT and DELETE[1)]. The REST handler must have the specific function for specific resource to handle those request methods.
According to the REST principle, GET method is used to retrieve a representation of resource from the server, POST method is used to create a new resource, PUT method for editing and DELETE method for deleting a specific resource.

Rest Support in Zend Framework

The Zend Framework 1.9 and later version has new features called **Zend_Rest_Controller** and **Zend_Rest_Route** which support RESTful application in Zend Framework[2)].

Zend_Rest_Route will route the HTTP request by translating the HTTP method and the URI to the appropriate module, controller and action.

Zend_Rest_Controller is the Zend controller which is used to map the HTTP method routed from Zend_Rest_Route into the specific action. It is an abstract class with 5 functions to process the REST methods: indexAction, getAction, postAction, putAction, and deleteAction. Extending the controller from Zend_Rest_Controller will help the use of Zend_Rest_Route.

2.1.3 Conclusion

Building a new REST handler would require more time to build the REST API then using the existing REST support in Zend Framework. However, the author will be able to easily use that handler because he know exactly the code. The REST support in Zend Framework also provide a very convenient way of building a REST API and it would require less time to build the REST API than build the code from zero.

As an open source code, the REST support in Zend Framework can be considered robust as it already published and used by many web developers, however, bugs might occur during the development process if the new REST handler is used, therefore more time might be required to fix the bugs.

Using the existing REST features in Zend Framework is the ideal solution for building the REST API within Zend Framework.

2.2 Which Data Model should be used for the API?

As the API will be implementing the REST principles, any data published by the API is only the representation of the resources. There are two types of commonly used data model for REST API, **XML** and **JSON**[1)].

2.2.1 Criteria

The ideal data model should have the following criteria:

- Well structured
- Self describing
- Standardized
- Extensible for future use
- Suitable for publishing web resources

2.2.2 Analysis

XML and JSON are very popular data model and has been used by many people nowadays. However, both of them have their own benefit and weakness.

XML

XML is an open standards set of rules for encoding documents electronically. It is a

textual data format, with strong support via Unicode for the languages of the world. Although XML's design focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services[3]. RSS, Atom, SOAP, and XHTML are example of XML-based languages that are widely used all over the world. XML is a better document exchange format[4]. XML stands for **Extensible Markup Language** which means it has the ability to extends it's format in the future.

One of XML data format which meets the criteria above is Activity Streams[6]. Activity Streams is an open standard data format to express social activities around the web. It is an extensions of Atom[5] which is an XML based language used for publishing web resources as a feed.

JSON

JSON is a lightweight data-interchange format. It uses the text format convention that are familiar to programmers of C, C++, C#, Java, JavaScript, Perl, Python, and many others. One of the advantage of using JSON is the relatively quick serialization time of data because the encoding mechanism of JSON adds only a minimum number of characters to indicate the structure and value of the data. JSON is a better data exchange format[7].

2.2.3 Conclusion

Both Activity Streams and JSON are well structured and standardized data format, however the Activity Streams has the benefit that JSON doesn't have. Activity Streams is a self describing document which is suitable for publishing web resources where JSON doesn't have an open standard format for it.

Although JSON has smaller data size than Activity Streams or any other XML formated documents, Activity Streams has the ability to describe the published resources more clearly.

Activity Streams will be the data model used for the API, because it meets the given criteria for the API's data model.

2.3 What Kind of Access Protocol Should be Used for the API?

The REST API must have an access protocol to be able to communicate properly with the third party application. Creating a new simple proprietary access protocol for Betavine Chronicle is one of the options, however there is another existing protocol called AtomPub[8] that can be implemented for the API as well.

2.3.1 Criteria

The ideal criterions for the access protocol are:

- Ease of implementation for the server
- Ease of implementation for the client
- Clear specification

– Open standard

2.3.2 Analysis

Simple Proprietary Access Protocol for Betavine Chronicle

This kind of protocol meets the first three criteria, which is, easy implementation in the server, easy implementation in the client, and might have a clear specification. However, designing a new protocol will require a lot of time before it can be implemented.

AtomPub

Atom Publishing Protocol is a well defined open standard REST based protocol for publishing, creating, and editing web resources. A clear specification of this protocol can be found in the internet[8)]. Since the specification is defined already, it should be easy for developers to implement the protocol, especially for those who already implement the protocol for another application, it might only take a few hours to implement the same protocol for the new application.

2.3.3 Conclusion

Both alternatives can be implemented easily either in the server side or in the client side. A clear specification can also be provided by both of them, however, a simple proprietary protocol can never be an open standard protocol like AtomPub.

Since Activity Streams has been chosen as the data model, AtomPub is the most suitable protocol, because AtomPub deals with atom format which is the basic format of Activity Streams. AtomPub will be the access protocol for the API and it will be publishing activity feed which is the Activity Streams formatted feed.

2.4 What Kind of XML Parser should be used for the API?

Since Activity Streams is chosen as the data model, the system needs an XML parser to parse any and build any XML document and to process it properly. SimpleXMLElement, Zend_Feed, and XMLReader are the alternatives available. The analysis for each parser will be explained after the criteria below.

2.4.1 Criteria

The XML parser for the API should meet the following criteria:

– ease of use

– ability to parse activity feed

2.4.2 Analysis

SimpleXMLElement

A very simple easily usable toolset to convert XML to an object that can be processed with normal property selectors and array iterators[9)].

Zend_Feed

A Zend_Framework feature which provides functionality for consuming RSS and Atom feeds. It also has extensive support for modifying feed and entry structure and turning the result back into XML[10)].

XMLReader

An XML Pull parser which acts as a cursor going forward on the document stream and stopping at each node on the way[11)].

2.4.3 Conclusion

Zend_Feed provides a very simple way to consume RSS and Atom feeds, however unnecessary operation might be executed, as the API will be only using Atom. At the moment, Zend_Feed doesn't have the capability to read activity feed, therefore Zend_Feed will not be used.

SimpleXMLElement provides an easy way to parse any XML document into a DOM object. It also provide a simple way to build an XML document.

As an XML pull parser, XMLReader requires less memory usage while parsing the XML document, however the parsing process is not as simple as SimpleXMLElement, therefore SimpleXMLElement was chosen to be the XML parser for Betavine Chronicle.

3. Proof of Concept (PoC)

In order to obtain the general knowledge about how to build the API and to estimate whether this project is feasible or not, the author built the PoC. It is a simple application that implements the basic techniques that would also be used to build the real API. PoC was built after conducting the research. It was implementing the results of the research.

The PoC has the following key points:

1) Zend Framework quickstart project.
2) Zend_Rest_Route and Zend_Rest_Controller.
3) Atom data format.
4) SimpleXMLElement.

To get acquainted with Zend Framework, the PoC was built from the quickstart tutorial provided by Zend Framework website[12)]. The Zend Framework quickstart project is a simple guestbook application which enable users to post a simple comment to the application. The goal of the PoC was to enable the third party application to see the last 10 comments and to post a new comment to the quickstart application.

The REST API was created using the Zend_Rest_Route and Zend_Rest_Controller. In this application only GET and POST requests were used. A simple Atom was used to represent the data (resources), therefore author created a simple Atom parser class using SimpleXMLElement.

The following code is used to initiate the Zend_Rest_Route in the application. It is located on the Bootstrap.php file.

```
protected function _initRestRoute()
{
    $frontController = Zend_Controller_Front::getInstance();
    $restRoute = new Zend_Rest_Route($frontController, array(), array('default' => array('api')));
    $frontController->getRouter()->addRoute('restapi', $restRoute);
}
```

Every Zend_Rest_Controller has at least 5 functions (indexAction(), getAction(), postAction(), putAction(), deleteAction()) where each function will process specific request method. Below is the Zend_Rest_Controller used in the PoC.

```
1  <?php
2  class ApiController extends Zend_Rest_Controller
3  {
4
5      public function init()
11     public function indexAction()
19     public function getAction()
73     public function postAction()
123    public function putAction()
130    public function deleteAction()
136 }
```

Example of Atom entry used for the PoC is shown below.

```
<entry>
  <id>id of guestbook #1 last items</id>
  <title>the #33 first item</title>
  <updated>2002-02-02T02:02:02+01:00</updated>
  <author>
    <email>aya_third_released@hotmail.com</email>
  </author>
  <content>2New status update...77dsf</content>
</entry>
```

The following is the Atom parser class built during this feasibility study. It is a simple Atom parser using SimpleXmlElement that can only read and write basic Atom document.

```
<?php
class Application_Model_AtomNode
{
    protected $_name;
    protected $_is_root;
    protected $_value;
    protected $_attributes;
    protected $_namespaces;
    protected $_namespace;
    protected $_childs;

    public function __construct($data,$is_root=true,$namespace='')
     * @param SimpleXMLElement $xml
    public function setNode(SimpleXMLElement $xml)
    public function addAttributes($attributes)
    public function addAttribute($name,$value)
    public function addChild($name,$value=null,$namespace=null)
    public function setElements($elements,$namespace)
    public function getValue()
    public function addNamespaces($namespaces)

    public function addNamespace($prefix,$namespace)
    public function getChilds()
    public function __get($name)
    public function toString($level=1)
    function setName($name)
    function setValue($value)
    function setIsRoot($isRoot)
    function setNamespace($namespace)
```

The code below is used to create an Atom entry response. It is a response for POST method request.

```
$entry = new Application_Model_AtomNode('<entry></entry>');
$entry->addNamespace('','http://www.w3.org/2005/Atom');
$entry->addChild('id','the #'.$comments[0]->getId().' item');
$entry->addChild('title','the #'.$comments[0]->getId().' item');
$entry->addChild('updated',atomnode_date_to_atom($comments[0]->getCreated()));
$Child = $entry->addChild('author',null);
$Child->addChild('email',$comments[0]->getEmail());
$entry->addChild('content',$comments[0]->getComment());

$this->getResponse()->setHttpResponseCode(201)->appendBody($entry->toString());
```

4. Conclusion

The PoC was tested using a simple mobile client application built by the other trainee. However, it cannot be tested while the other trainee is still working on it. No unit testing was built, the only way to test the PoC was by using that mobile client application. Therefore, a unit testing should be built for the API to break that dependency. The unit testing will imitate the client request and test whether the response is correct or not, thus no client application is needed to test the API.

The simple atom parser created during the building of PoC can read and write any simple atom document very well, however Activity Streams will be used for the project, therefore a different parser should be built in the implementation phase. Based on the research's result, SimpleXMLElement will be used to build the Activity Streams parser.

The PoC prove that the author have learned the required skill and knowledge regarding to the project, therefore the author would be able to finish this project.

References

1) http://en.wikipedia.org/wiki/Representational_State_Transfer
2) http://devzone.zend.com/article/4906
3) http://en.wikipedia.org/wiki/XML
4) http://www.json.org/xml.html
5) http://en.wikipedia.org/wiki/Atom_(standard)
6) http://activitystrea.ms
7) http://www.json.org/index.html
8) http://bitworking.org/projects/atom/rfc5023.html
9) http://www.php.net/manual/en/intro.simplexml.php
10) http://framework.zend.com/manual/en/zend.feed.introduction.html
11) http://www.php.net/manual/en/intro.xmlreader.php
12) http://framework.zend.com/manual/en/learning.quickstart.html

Functional Requirement Document
Fontys University of Applied Sciences

"Open The Chronicle"

Author	:	Eko Adi Prasetyo Gunawan
Student Number	:	2154590
School Tutor	:	Erik H.J.D. van der Schriek
Company Tutor	:	Laurent Eschenauer
Version	:	0.3
Date	:	05/29/10

Version Control

VERSION	DATE	DESCRIPTION	REASON
0.1	Mar 25, 2010	Initial version	
0.2	Apr 12, 2010	1. Requirement F14 added. 2. Requirement F7f become 'Should have'.	1. New Requirement. 2. Mistake on version 0.1.
0.3	May 29, 2010	1. Report's appearance changed. 2. Version control added.	1. To Follow the appearance of Final Report. 2. To track the changes in each version.

Table of Contents

1. Introduction

The “API for Betavine Chronicle” (ABC) project's main goal is to create a REST API which enable the third party application to do some functionality in Betavine Chronicle. The detailed description of the project can be seen in the Project Plan. The client has already given a list of requirements for the project. That requirement list will be divided based on its priorities in the MoSCoW table. Requirements which are categorized as “Must Have” have to be implemented for the project to be considered succeed, whereas the other requirements will be implemented if there is enough time in the development process. This document will be updated if there are new requirements added into the project.

2. Functional Requirement

- **REST API Functionality:**

  F1. Post a new item.

  - Process any client's HTTP POST request to create a new item and response with the representation of that item.

  F2. Edit an existing item.

  - Process any client's HTTP PUT request to edit an item and response with the representation of the updated item.

  F3. Retrieve an item.

  - Process any client's HTTP GET request to retrieve an item and response with the representation of the item.

  F4. Retrieve a list of items.

  - Process any client' HTTP GET request to retrieve a list of items and response with the representation of the item list.

  F5. Delete an item.

  - Process any client's HTTP DELETE request to delete an item.

  F6. Show/hide an item from public timeline.

  - Enable the client to change the privacy of an item.

  F7. Support the following item types.

  - **F7a. Status.**
    - A user's status update.
  - **F7b. Blog.**
    - A user's blog post consist of title and content.
  - **F7c. Picture.**
    - A user's uploaded picture (uploaded in Chronicle or a link of picture hosted somewhere else) with title and note.
  - **F7d. Link.**
    - A user's shared link consists of the link, title and note.
  - **F7e. Audio.**
    - A user's uploaded audio file (uploaded in Chronicle or a link of audio file hosted somewhere else) with title and note.

- **F7f. Video.**
  - A user's embedded code of video (hosted somewhere else) with title and note.

F8. Retrieve latest activities of a user.

- Process any client's HTTP GET request to retrieve latest activities (update status, post a blog, shared a link, etc..) of the user and response with the representation of the activities.

F9. Post a comment on an item.

- Process any client's HTTP POST request to add a new comment on an item and response with the representation of that comment.

F10. View comments of an item.

- Process any client's HTTP GET request to retrieve a comment list of an item and response with the representation of the comment list.

F11. Edit a comment.

- Process any client's HTTP PUT request to edit a comment of an item and response with the representation of the updated comment.

F12. Delete a comment.

- Process any client's HTTP DELETE request to a comment of an item.

F13. Importing external activity streams.

- Import any activity streams from external sources into Betavine Chronicle.

F14. Building an Atom Adapter Library

- A library to read and write any Atom based documents with different kind of extensions. It is also expected to be published as an open source library.

3. Non-Functional Requirement

- **N1. API Documentation**
  - A document that describe the specification of the API to help the third party developers to communicate with the API.
- **N2. Security**
  - Provide a more secure way of communication with the third party client using OAuth for the authentication.
- **N3. Quality**
  - Provide the API with the best quality in term of less bug by creating a unit testing for each functionality to check whether it works correctly.
- **N4. Scalability**
  - Provide a robust API which can handle a growing amount of client that communicates with the API in the future.
- **N5. Google PUSH (PubSubHubBub[9]) Implementation.**
  - Provide a PUSH mechanism of publishing the web resources using PubSubHubBub which enable the client to have a near-instant notification of a newly updated or newly created web resource.

[9] http://code.google.com/p/pubsubhubbub/

4. MoSCoW Table

ID	Requirement	M	S	C	W
F1	Post a new item	√			
F2	Edit an existing item	√			
F3	Retrieve an item	√			
F4	Retrieve a list of items	√			
F5	Delete an item	√			
F6	Show/hide an item from public timeline		√		
F7a	API functionalities for Status item types	√			
F7b	API functionalities for Blog item types	√			
F7c	API functionalities for Picture item types		√		
F7d	API functionalities for Link item types	√			
F7e	API functionalities for Audio item types		√		
F7f	API functionalities for Video item types		√		
F8	Retrieve latest activities of a user	√			
F9	Post a comment on an item		√		
F10	View comments of an item		√		
F11	Edit a comment		√		
F12	Delete a comment		√		
F13	Importing external activity streams			√	
F14	Building an Atom Adapter Library	√			
N1	API Documentation				
N2	Security		√		
N3	Quality	√			
N4	Scalability		√		
N5	Google PUSH implementation			√	

Note:
M - MUST have this.
S - SHOULD have this if at all possible.
C - COULD have this if it does not affect anything else.
W - WON'T have this time but WOULD like in the future.

API Documentation

Fontys University of Applied Sciences

"Open The Chronicle"

Author	:	Eko Adi Prasetyo Gunawan
Student Number	:	2154590
School Tutor	:	Erik H.J.D. van der Schriek
Company Tutor	:	Laurent Eschenauer
Version	:	0.1
Date	:	Jun 09, 2010

Version Control

VERSION	DATE	DESCRIPTION	REASON
0.1	Feb 17, 2010	Initial version	
0.2	Jun 30, 2010	Changes on Chronicle API specification	New version of the Chronicle API

Table of Contents

1. Introduction

This document is intended for developers who want to write applications that can interact with the Betavine Chronicle API. Betavine Chronicle is a tool for sharing status updates, blog post, links, photos, audios and videos. The Betavine Chronicle API provides functionalities to create, update, and delete resources from Betavine Chronicle. This document assumes that the readers are familiar with web programming concepts and web data formats.

2. Chronicle API Resources

A resource is an individual data entity with a unique identifier. Resources in Betavine Chronicle represent the activities of its user. There are several types of resource in Betavine Chronicle:

- Status updates;
- Blog posts;
- Pictures shares;
- Link shares;
- Audio shares;
- Videos shares.

The list of resources mentioned above is called activity collection. It is a list of the latest activities of a specific user.

3. Chronicle API Operations

Operation	Description	REST HTTP Mappings
List	Lists all resources within a collection.	GET on a collection URI.
Insert	Inserts a new resource into a collection.	POST on a collection URI. (with data)
Get	Gets a specific resource.	GET on resource URI.
Update	Updates a specific resource.	PUT on resource URI.
Delete	Deletes a specific resource.	DELETE on resource URI.

Each operation will return an HTTP response code with the following specification:

- For all failed requests, the Chronicle API will return **400;**
- For successful POST request, the Chronicle API will return **201**;
- For successful GET, PUT, and DELETE request, the Chronicle API will return **200**.

4. Data Format

The data format used to represent the activity resources in Betavine Chronicle is the

Activity Streams. It is an extension to the Atom syndication format to express what people are doing around the web. To help understanding the concept of Activity Streams let us review the Atom syndication format.

4.1 Atom Syndication Format

An XML-based document format that describes lists of related information as "feeds". It is an open standard format that primarily used for web content syndication. All elements within Atom document must be in the http://www.w3.org/2005/Atom namespace.

Atom document is divided into two main type: Atom Feed document and Atom Entry document. Atom Entry document represents an individual entry, acting as a container for metadata and data associated with the entry. Atom Entry contains several metadata elements such as atom:id, atom:title, atom:updated, atom:author, atom:link, atom:content, etc.. Atom Feed document acting as a container for metadata and data associated with the feed, its element children consist of metadata elements followed by zero or more atom:entry child elements. Please refer to the Atom Syndication Format specification[10] to read more detail about Atom.

4.2 Activity Streams

As an extension of Atom document, Activity Streams add more information that allows activities on social objects to be expressed within the Atom Syndication Format. Several namespace are used in Activity Streams document, for example: http://activitystrea.ms/spec/1.0/, http://purl.org/syndication/thread/1.0, http://purl.org/syndication/atommedia , etc..

An activity is a description of an action that was performed (the verb) at some instant in time by some actor (the actor), with some social object (the object). An activity feed is a feed of such activities. An activity may also have a target, which is the object into which or to which the action was done. Whether a target is allowed and the specific meaning of a target is defined by each verb. It is expected that in many cases consumers of activity feeds will use them to turn machine readable description of activities into human-readable sentences such as "Joanne posted a Photo: 'My Cat'".

Please read the Atom Activity Extensions specification[11] to understand the structure of Activity Streams document.

[10] http://tools.ietf.org/html/rfc4287
[11] http://activitystrea.ms/spec/1.0/atom-activity-01.html

5. Working with the Chronicle API

This chapter is explaining how to communicate with the Chronicle API. Several example are given to make it easier to be understood. The examples below only use the one type of activity resources; however, there are other types of activity resources which are supported by Betavine Chronicle as mentioned in the **chapter 2** of this document. To see the activity entry's specification of each type please see the *Atom Activity Base Schema*[12].

5.1 Authentication

The Chronicle API uses basic authentication. To be able to use some methods from the Chronicle API, third party application must provide the username and the password of the user. Add a username parameter to every endpoint URL, and a 'password' header with the user's password in the request header. The password is required for POST, PUT, and DELETE methods.

Example:

The following is an example of the url and the request header.

URL: chronicle.loc/api/activities?username=ekoadipg

```
POST /api/activities HTTP/1.1
Host          : chronicle.loc
Password      : storytlr
Content-Type  : application/atom+xml
```

5.2 Activity Feed

This provides all of the public activities posted by a specific user. This method doesn't require a password authentication. To access this feed, send an HTTP GET request to the following endpoint URL:

http://chronicle.loc/api/activities?username=*the-username*

Response

If the request succeeds, the serves responds with an HTTP 200 status code and an activity feed of the latest activities of a specific user. For example:

[12] http://activitystrea.ms/schema/1.0/activity-schema-01.html

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:activity="http://activitystrea.ms/spec/1.0/"
      xmlns:media="http://purl.org/syndication/atommedia">
    <title>ekoadipg's activities</title>
    <id>id_ekoadipg_3</id>
    <updated>2010-05-08T12:26:30+02:00</updated>
    <author>
        <name>ekoadipg</name>
        <uri>http://ekoadipg.chronicle.loc</uri>
        <email>ekoadipg@hotmail.com</email>
    </author>
    <link rel="self"
          href="http://chronicle.loc/api/activities?username=ekoadipg"/>
    <entry>
        <id>4_173_status</id>
        <title>Example of a status update</title>
        <updated>2010-05-07T14:52:00+02:00</updated>
        <link rel="alternate"
              href="http://ekoadipg.chronicle.loc/entry/example-of-a-status-update-4-173.html"/>
        <link rel="edit"
              href="http://chronicle.loc/api/activities/4_173_status?username=ekoadipg"/>
        <content type="html">Example of a status update</content>
        <activity:verb>http://activitystrea.ms/schema/1.0/post</activity:verb>
        <activity:object>
            <activity:object-type>http://activitystrea.ms/schema/1.0/status</activity:object-type>
            <id>4_173_status</id>
            <title type="text">Example of a status update</title>
            <link rel="alternate"
                  type="text/html"
                  href="http://ekoadipg.chronicle.loc/entry/example-of-a-status-update-4-173.html"/>
            <content type="html">Example of a status update</content>
        </activity:object>
    </entry>
    <!-- more entry -->
</feed>
```

Notice that each entry has a link with a rel value of “edit”. It is the endpoint of the activity where we can send an HTTP request to that specific activity.

5.3 *Create A New Activity*

To post a new activity for a user, send an HTTP POST request to the user's activity feed end point. For posting a media activity, e.g. upload picture, the request body must contain the media binary data, otherwise an activity entry must be sent as the body of the request. This method requires a password authentication.

http://chronicle.loc/api/activities?username=*the-username*

Request

The body of the request contains the content of the new post or a media file. For example:

```
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:activity="http://activitystrea.ms/spec/1.0">
    <activity:verb>http://activitystrea.ms/schema/1.0/post</activity:verb>
    <activity:object>
        <activity:object-type>http://activitystrea.ms/schema/1.0/status</activity:object-type>
        <content type="html">Example of a status update</content>
    </activity:object>
</entry>
```

Note:

1. No need to supply the “atom:id” or “atom:author” elements; the server creates those in response to the POST request. These element will be ignored if present.

2. If the “atom:updated” or “atom:published” element is present, it will be used as the published date of the activity, otherwise the current date and time will used.

Response

If the request succeeds, the server responds with an HTTP 201 status code and the full entry that was created. For example:

```
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:activity="http://activitystrea.ms/spec/1.0">
    <id>4_173_status</id>
    <title>Example of a status update</title>
    <updated>2010-05-07T14:52:00+02:00</updated>
    <link rel="alternate"
          href="http://ekoadipg.chronicle.loc/entry/example-of-a-status-update-4-173.html"/>
    <link rel="edit"
          href="http://chronicle.loc/api/activities/4_173_status?username=ekoadipg"/>
    <content type="html">Example of a status update</content>
    <activity:verb>http://activitystrea.ms/schema/1.0/post</activity:verb>
    <activity:object>
        <activity:object-type>http://activitystrea.ms/schema/1.0/status</activity:object-type>
        <id>4_173_status</id>
        <title type="text">Example of a status update</title>
        <link rel="alternate"
              type="text/html"
              href="http://ekoadipg.chronicle.loc/entry/example-of-a-status-update-4-173.html"/>
        <content type="html">Example of a status update</content>
    </activity:object>
</entry>
```

Notice that the new entry also has a link with a rel value of “edit”. As mentioned before, it is the endpoint of the activity where we can send an HTTP request to that specific resource.

5.4 Read an Activity

To read an individual activity, send an HTTP GET request to the endpoint which is obtained from the activity feed or the entry response of a newly created activity. This method doesn't require a password authentication. For example:

http://chronicle.loc/api/activities/4_173_status?username=ekoadipg

Response

The response is an entry of that activity, the same response as the newly created entry after posting a new activity.

5.5 Update an Activity

To update an activity, send an HTTP PUT request to the same endpoint as reading the activity. This method require a password authentication. For example:

http://chronicle.loc/api/activities/4_173_status?username=ekoadipg

Request

The body of the request contains the full updated content of the activity. For example:

```
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:activity="http://activitystrea.ms/spec/1.0">
    <id>4_173_status</id>
    <title>Updated: Example of a status update</title>
    <updated>2010-05-07T14:52:00+02:00</updated>
    <link rel="alternate"
          href="http://ekoadipg.chronicle.loc/entry/example-of-a-status-update-4-173.html"/>
    <link rel="edit"
          href="http://chronicle.loc/api/activities/4_173_status?username=ekoadipg"/>
    <content type="html">Updated: Example of a status update</content>
    <activity:verb>http://activitystrea.ms/schema/1.0/post</activity:verb>
    <activity:object>
        <activity:object-type>http://activitystrea.ms/schema/1.0/status</activity:object-type>
        <id>4_173_status</id>
        <title type="text">Updated: Example of a status update</title>
        <link rel="alternate"
              type="text/html"
              href="http://ekoadipg.chronicle.loc/entry/example-of-a-status-update-4-173.html"/>
        <content type="html">Updated: Example of a status update</content>
    </activity:object>
</entry>
```

Response

If the request succeeds, the server responds with an HTTP 200 status code and the full updated entry (the same as the request body).

5.6 Delete an Activity

To delete an activity, send an HTTP DELETE request to the same endpoint as reading the activity. This method requires a password authentication. For example:

http://chronicle.loc/api/activities/4_173_status?username=ekoadipg

No request body is required in this request. If the request succeeds, the server responds with an HTTP 200 status code.