# COMPSCI 335 Assignment 2

*A Community Blogger REST Web Service*

COMPSCI 335 Assignment 2

# COMPSCI 335 Assignment 2

*A Community Blogger REST Web Service*

The purpose of this assignment is for you to demonstrate that you can build a REST-based Web service. You should develop the Web service using the JAX-RS and JAXB specifications, and use Maven to manage the build process.

The Web service is a Community Blogger service that is to manage a collection of blog entries. Registered users of the service can post blog entries that include text content and optionally a set of keywords. Each registered user is uniquely identified by their username. In addition to handling new blog posts, the Web service should allow registered users to comment on particular posts. The Web service should also process query requests to retrieve blog entries, details of registered users and any comments associated with a blog entry. Finally, the Web service should allow one user to 'follow' a specified blog entry, such that the user is notified whenever the entry is commented on.

This assignment is worth 6% of your COMPSCI 335 mark.

**NOTE**: do not attempt this assignment until you have worked through the practical introduction to using Maven. This is supported by a video; both the practical exercise and video are available on Piazza.

## Resources

To help you develop the Web service, a number of resources are provided.

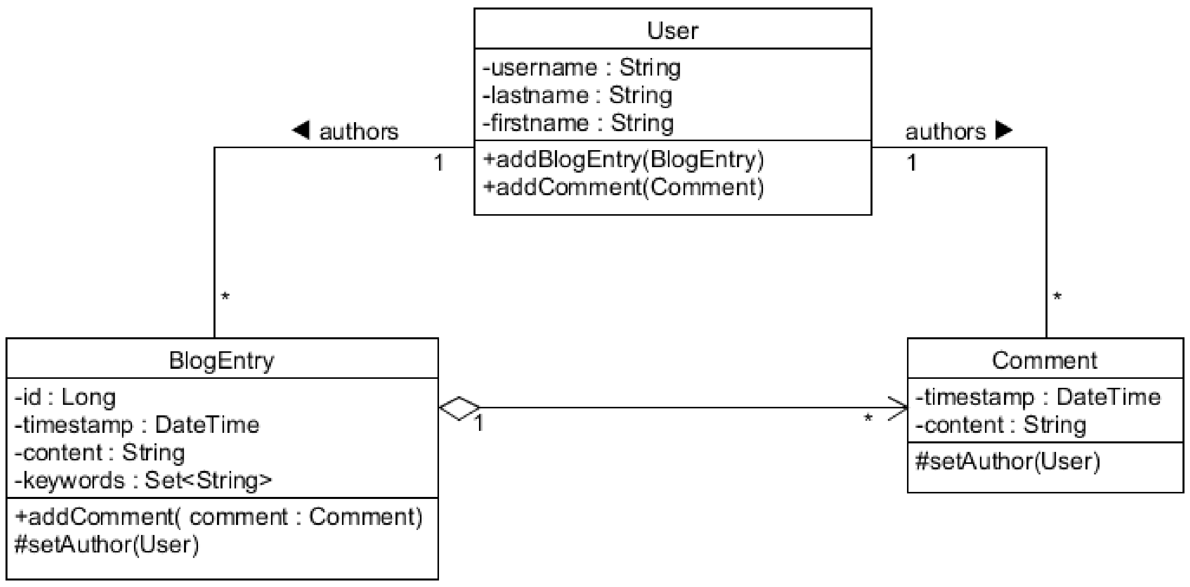
- ***Domain model***. The UML class diagram on page 2 represents the Community Blogger's domain model. The `BlogEntry`, `User` and `Comment` classes represent key problem domain elements.

- ***Service contract***. The service contract specifies the operations of the Community Blogger Web service. Your service should conform to the specification.

- ***Skeleton project***. This is available on Piazza and includes an implementation of the domain model, a skeletal Web service interface with partial implementation, and a POM file that contains the dependencies you are likely to need along with plugin configuration.

## Domain model

The following class diagram shows the domain model classes and relationships.



Note that the association between `User` and `BlogEntry` is bidirectional, as is the association between `User` and `Comment`. This means that given a `User` object it's possible to retrieve its `BlogEntry` and `Comment` instances, and that with a `BlogEntry` or `Comment` object, it's possible to navigate back to the `User` object.

To illustrate use of the model, consider the blogging activity:

**Post**: Maven is a cool tool for building Java projects! …

By Dave at 5:20pm on 15 August 2015

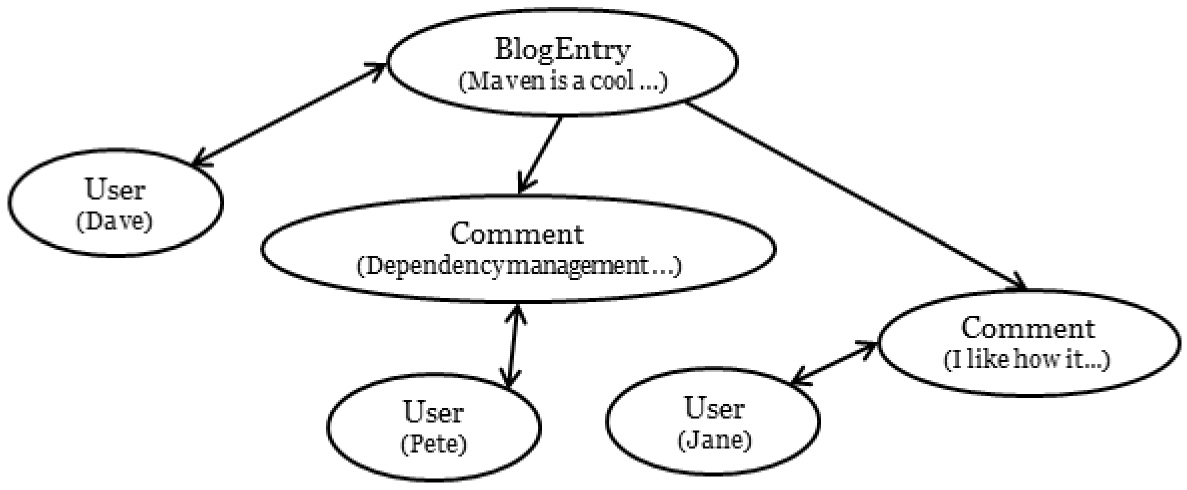
**Comment**: Dependency management is an awesome feature!

By Pete at 6:36pm on 15 August 2015

**Comment**: I like how it simplifies testing of Web services.

By Jane at 2:14pm on 16 August 2015

To represent this activity, instances of the domain model classes would be constructed and linked as shown over-page.



## Service contract

The Community Blogger Web service must offer the following operations:

***Create user*.** Attempts to create a new user with a specified username. If the supplied username is already taken, the request fails and the Web service doesn't change its state. In other cases, where the username is new (unique), the Web service updates its state and stores the new user.

*Request input*: The HTTP request body stores a XML representation of a user. The representation must include the following attributes: username, last name and first name.

*Response output*: the HTTP response body is empty. If the request is successful, the HTTP response code is 201 (Created) and a Link header is included with a URI to identify the newly created user. For unsuccessful requests a 409 (Conflict) response code is returned.

***Retrieve user*.** Retrieves a user by their username. This operation is successful if the username specified in the request identifies a user that is stored by the Web service.

*Request input*: the request URI should include the username of the user to be retrieved.

*Response output*: for successful executions, this operations returns a response code of 200 (OK) and the response body contains an XML representation of the user. This must include the user's username, last name and first name. Where the request fails, the response code is 404 (Not Found) and the response body is empty.

***Create blog entry*.** Creates a new blog entry. This operation is successful only when the request includes a cookie header named "username" and where the cookie value identifies a user that is stored by the Web service. Successful execution involves storing a new blog entry in the Web service. In the case of failure, the Web service's state is unchanged.

*Request input*: a XML representation of a new blog entry. A mandatory element is the blog entry's content. Keywords are optional and may or may not be included. A timestamp value recording the blog entry's creation time may be supplied, but the Web service will ignore this and use the server's clock to record the blog entry creation time. The blog entry's author needn't be included in the XML representation, because the required "username" cookie's value should be used for this.

*Response output*: for successful operation, the response code is 201 (Created) and a Link header is returned with a URI identifying the newly created blog entry. The URI includes a unique ID value for the blog entry that the Web service assigns execution of the request. To indicate failure, a response code of 412 (Precondition Failed) is returned.

***Retrieve blog entry*.** Retrieves a particular blog entry by its unique ID. This operation is successful when the specified ID identifies a blog entry that is stored by the Web service.

*Request input*: the request URI should include the unique ID for the blog entry to retrieve.

*Response output*: where successful, this operation returns a 200 (OK) response code and the response body includes an XML representation of the blog entry. This must include the blog entry's author (username), content, any keywords, and the entry's timestamp. The XML representation shouldn't include comments or other information about its author. In failure cases, this method returns 404 (Not Found) and an empty response body.

***Create comment*.** Creates a new comment for a specified blog entry. This operation is successful only when the following conditions are satisfied:

- The request includes a cookie header named "username" where the cookie value identifies a user that is stored by the Web service. The value of this cookie identifies the author of the comment.
- The request identifies a blog entry that is stored by the Web service.

*Request input*: the request URI should include the unique ID for the blog entry for which a new comment is to be posted. As described above, the request should also include the required cookie. The request body should include a XML representation of the comment, with elements to include the comment's content. The XML needn't include a timestamp for the comment's creation date/time (the Web service will generate this value when storing the new comment). Similarly, a username element to identify the author isn't mandatory; the Web service should use the cookie value to identify the author.

*Response output*: for successful operation, a 200 (OK) response code is returned. For failure cases, a 412 (Precondition Failed) code is returned if the required cookie is absent or its value doesn't identify a registered user. Where the cookie is valid but the blog entry fails to match a known blog entry, the Web service returns a 404 response.

***Retrieve comments***. Retrieves a set of comments for a specified blog entry. This operation is successful when the ID given for the blog entry identifies a blog entry that is known to the Web service.

*Request input*: the URI should include the ID of the blog entry for which comments should be retrieved.

*Response output*: in successful cases, this operation returns a response code of 200 (OK). The response body contains an XML representation of a set (that may be empty) of comments. Each comment should include the username of the comment's author, the comment's content and timestamp. Where the request fails, a response code of 404 (Not found) is returned.

***Retrieve blog entries***. Retrieves a set of blog entries.

*Request input*: none.

*Request output*: an XML representation of a set of blog entries stored by the Web service. Where the service doesn't contain any entries, the returned set is empty.

A simple implementation of this operation will return all blog entries that are stored by the Web service. Better implementations will support the following enhancements:

- Ensuring that there is an upper bound on the number of blog entries returned from any one invocation.
- Allowing clients to express query options, such that only blog entries matching particular criteria are returned. Possible

criteria include blog authors, blog ID ranges, timestamp ranges, keywords etc.

***Follow blog entry***. Follows a particular blog entry. Whenever a comment is posted on the blog entry, the client that makes this request is notified asynchronously. If the specified entry-to-follow doesn't exist in the Web service, this operation aborts immediately; otherwise, it waits for a new comment to be posted and then returns, notifying the requesting client.

*Request input*: the URI embeds the blog entry ID of the blog entry to follow.

*Request output*: an XML representation of any new comment(s) that have been posted on the blog entry being followed. A comment should be described by its author's username, its content and timestamp.

A good implementation of this operation will guarantee that clients do not miss new comments that are posted between the client's subscription renewal requests (i.e. calls to this operation).

The service contract distinguishes between *registered* and *anonymous* users. A registered user is one who is known to the Web service, and who has a unique username. Only registered users can post blog entries and comments. Anonymous users can register by invoking the service's ***create user*** operation.

It is of course possible for any client request to claim to be from a particular registered user. In reality, a secure means of interacting with the Web service would be required. We won't address security in part 2 of COMPSCI 335, but part 3 will return to this topic.

## Suggested tasks

To progress through the assignment, we suggest that you work through the following tasks in the order presented:

1. Design the REST-based interface for the Community Blogger Web service. You should study the service contract and design a URI scheme and associate each URI with a particular HTTP method. Each operation of the service contract requires a URI and HTTP method. For operations that exchange data between a client and the Web service, you will need to consider whether to exchange domain objects or introduce DTOs.

2. Implement a resource class for the Web service. This class should include the Service's state (users, blog entries and comments) and handler methods, one for each operation. You should use JAX-RS annotations appropriately to implement the resource class. Similarly you'll need to use JAXB annotations on classes whose objects are to be exchanged between clients and the Web service. To start with, consider a simple

implementation of operation ***retrieve blog entries*** (one that doesn't limit the number of blog entries to return and which doesn't support search criteria). Also, don't be concerned with operation ***follow blog entry*** – leave this until you've got the rest of the service implemented.

3. Implement the remaining classes that are required for a JAX-RS Web service. Essentially, these are the Application subclass, that is used to register singleton objects (an instance of your resource class) and component classes (e.g. a context resolver to configure JAXB for your application).

4. Write unit tests, as necessary, for each of the Web service operations. In general, you need 2-3 units tests for each operation to demonstrate that the operations behave according to the service contract.

5. Return to operation ***retrieve blog entries*** and enhance it as described in the service contract. You should aim to implement one technique for returning some subset of blog entries and at least one form of criteria-based querying.

6. Design and implement a solution for operation ***follow blog entry***. A simple implementation will allow a client to subscribe to future comments for a particular blog entry. A better solution will, as described in the service contract, will guarantee that a subscribing client will not miss any relevant comments that are posted in the window between the client receiving a notification and making a subsequent subscription. On re-subscribing, if further comments have been posted for the blog entry of interest, the client should be notified immediately of the new comments.

If you want to start from scratch, that's fine. Alternatively, you may want to use the supplied resources – the domain model and skeletal project – and build a solution that uses them. If using the provided domain model classes, refer to the Javadoc comments for further information on how to use them.

## Assessment criteria

Your work will be assessed according to the following criteria:

- ***Conformance to REST principles***. You solution should conform to the basic principles of REST and should not subvert the HTTP protocol (e.g. GET requests must not cause Web service state changes, in order for the results of a GET request to be safely cached).

- ***Service correctness***. Your unit tests should demonstrate that the Web service implementation meets the service contract, behaving as required in cases of normal and error conditions.

- ***Service functionality***. Your implementation should minimally satisfy the basic service contract (i.e. all operations with the exception of operation follow blog entry should be implemented, and retrieve blog entries can be implemented to return all known blog entries). Extra credit will be given for functional enhancements.

- ***Soundness of implementation***. Your implementation need only be tested with requests arriving sequentially. However, the service implementation is intended for deployment in a multithreaded environment, following a servlet container's thread-pre-request model. Extra credit will be given for implementations that aren't susceptible to threading errors (e.g. data corruption, race conditions).

- ***Maven build***. The project's build must be driven by a Maven POM file. By running Maven install, the project must pass through the build cycle and run the unit tests against the Web service.

## Submission

You should submit **ONE** zip file to the Assignment Drop Box:

https://adb.auckland.ac.nz

The zip file should contain **ONE** project that is structured according to Maven's default directory structure. The Maven project should include all resources other than third party libraries that can be downloaded from the Maven central repository. During assessment, markers will build and run your project – a project that does not run using Maven's *install* target will be assumed to be non-functional.

The due date for this assignment is 6:00pm on Mon 21 September 2015.