RESTful APIs: A simple messaging application

Reading / References

- Restful Web Services[1], chapters 4 and 5. Optionally 6, 7, 8.
- Current HTTP Specification[2] for some "light reading".

Notes

We will consider here a simple messaging application. Users will be able to send "messages" to each other, and also "tag" them with keywords for later searches. We will keep the system simple for now and not discuss any security concerns. Our main steps in this design process would be the following:

1. Identify clients and their needs.
2. Identify the specific *resources* that would need to be addressable.
3. Describe the addressing scheme for each resource.
4. Describe the behavior of the different HTTP verbs on each resource.
5. Decide the "payload" required for the various requests as well as the payload returned from various requests.
6. Decide how the service could be *navigated*, and links between the various resources.
7. Figure out the error cases and how your system should behave in those cases.

Clients and their needs

The first step is to identify who would need to interact with our application, and what they want to be able to do. Here are some examples, for two specific clients (a user and an administrator):

1. A user needs to be able to create a new account.
2. A user needs to be able to compose and send a message to another user.
3. An administrator needs to be able to see a list of all the users and how many messages they each have.

Groupwork: Determine other actions that the various kinds of clients need to be able to take. You may include questions that you would like to have answered in order to clarify the situation more (for example, should users be able to save a draft message on the server, and only send it out later?).

[1] http://learning.acm.org/books/book_detail.cfm?id=1406352&type=safari
[2] https://tools.ietf.org/html/draft-ietf-httpbis-p2-semantics-21

Addressable resources

The next key step is to identify the resources that should have their own addressing scheme. Here is how the *Restful Web Services* book describes resources:

> Every interesting thing your application manages should be exposed as a resource. A resource can be anything a client might want to link to: a work of art, a piece of information, a physical object, a concept, or a grouping of references to other resources.

Groupwork: Think about our messaging application, and what our different resources might need to be.

Addressing Scheme

For each resource, we have to decide what the addressing scheme for that resource should be. There is often a tradeoff between how descriptive the scheme and how resilient to change it is, and a key question is how of a concern such a change might be. For example, suppose we wanted to represent course sections in our evaluation site. A course section might have one of the following example schemes:

```
/section/CS/220/A
/section/15624        <----- A unique ID generated for the section
```

So the first scheme follows a "department/number/section letter" format. Looking at that URL tells us almost exactly which section we are after (question: Why did we say "almost" there? What is missing from this scheme?). The second scheme is more obscure, using only a section ID.

Now which of these is better? There is no simple answer. The first scheme is certainly more human-readable, which is a nice perk. But imagine that we had to rename the sections, and this section became section B, and a new section A was created. Then the two links actually refer to different sections now! Or maybe we decided to change the department code from two letters to three "CSC". All the previous links would break and we might need to issue some permanent redirect instructions on them.

Groupwork: Think about reasonable addressing schemes for the resources in our messaging application, and any possible tradeoff concerns.

Verb behavior

For each of the addressing schemes, we need to decide which HTTP verbs would be allowed and what each verb is meant to do. For instance, on some resources perhaps a GET is the only meaningful verb, while for others perhaps POST is the main thing we'd want to do.

Groupwork: For each of the addressing schemes described earlier, identify what the various verbs should do when encountering those schemes.

Payload

Some operations require information in order to be carried out, while others return information. For example in order to send a message, we need to provide the text for the message, specify a recipient and a message subject etc. Or for example when we ask for a list of messages, something needs to be sent back. What is that something? Do we get the entire messages back? Maybe just their subjects/recipients? What about their tags? And so on.

Groupwork discuss the payload returned from the various methods as well as the payload required by the various methods.

Navigation

In order to discover what we can do with the service, we need a way to navigate it. This can often be done by having the payloads returns by the various operations contain links to other resources. For example when you access a user resource, you may receive back a "link" to the messages of that user, and perhaps a link back to the list of all users, and so on.

Groupwork discuss what various links might be applicable for our simple messaging application.

Error Handling

Finally, we must specify all the wrong behaviors and how we would react to each. For instance:

- Is there a limit to how large the message body or the message subject can be?
- What should happen to messages addressed to non-existent users?

Groupwork discuss other possible error cases and how our system should handle them.