KTH V17P01 Programming Web Services: Homework 3

Fannar Magnusson (fannar@kth.se)
Thorsteinn Thorri Sigurdsson (ttsi@kth.se)

February 13, 2017

1 Introduction

The assignment focused on giving hands on experience developing web services and web service client. This time we developed a RESTful Web Service with the help of Jersey and GlassFish to deploy the service.

2 Implementation

Our web service is split up to four resources: *users, login, itineraries and booking.*

2.1 Users resource

Has the path: `/users` and is responsible for everything regarding user management. Implements the methods `createUser` - which is a POST method that takes in Form parameters, `updateUser` - which is a PUT method that takes in a User JAXBElement and returns the updated user, `deleteUser` - DELETE (sent to `/users/{id}`) and `getUsers/s` - GET. It is possible to get a list of all users in the system by sending a GET request to `/users` or the details of a single user by sending a GET request to `/users/{id}`.

2.2 Login resource

Has the path: `/login`. Returns a User object. Authentication is handled in the request filter, see Authentication subsection, so this method simply returns the user object.

2.3 Itineraries resource

Has the path: `/itineraries`. It implements the `GET` method `findItineraries` which takes in three string input query parameters: departure, destination and date. The method finds all available itineraries Returns a list of itineraries that can be booked with the booking resource.

2.4 Booking resource

Has the path: `/booking`. Allows `POST` operations. Input is an `Itinerary` and `PaymentInfo`, wrapped together in a `BookingRequest` XML. The service then books and issues tickets for the flights in the requested itinerary, and returns the tickets.

2.5 Authentication

We use HTTP Basic Authentication for authentication. On the server side we inject a `ContainerRequestFilter` into the request handler chain. In this filter we fetch the authorization string from the HTTP headers, decode it and check if the encoded username and password are valid. If they are not, we throw a `401 Unauthorized` error.

An exception to this is the `POST` method to the `/users` endpoint. This is used to create a new user and does not request authorization.

The filter also constructs a `SecurityContext` object that gets attached to the `RequestContext` of the incoming request, and can be used later in the request handler chain, i.e. in the REST service itself, to identify the user that made the request.

On the client side we put in the appropriate authorization header, with the user's username and password, before making the request.