Chapter No.1
"Setting Up the Environment"
In this package, you will find:

The authors biography

A preview chapter from the book, Chapter no.1 "Setting Up the Environment"

A synopsis of the book’s content

Information on where to buy this book

About the Authors

René Enríquez is currently a software architect for a multinational company headquartered in India. He has previously worked on many projects related to security implementation using frameworks such as JAAS and Spring Security to integrate many platforms based on the Web, BPM, CMS, and web services for government and private sector companies. He is a technology and innovation enthusiast, and he is currently working with several programming languages. He has achieved the following certifications:

- Oracle Certified Professional, Java SE 6 Programmer
- Microsoft Technology Associate
- Cisco Network Operating Systems

Over the past few years, he has worked as a software consultant on various projects for private and government companies and as an instructor of courses to build enterprise and mobile applications. He is also an evangelist of best practices for application development and integration.

For More Information:
Andrés Salazar C. is currently working at one of the most prestigious government companies in Ecuador, performing tasks related to software development and security implementation based on JAAS and digital signatures for secure applications. He also has extensive knowledge of OAuth implementation on web projects. He is a technology and Agile enthusiast, and he has worked on several projects using the JEE technology and TDD. He has achieved the following certifications:

- Oracle Certified Professional, Java SE 6 Programmer
- Certified Scrum Developer

For More Information:
RESTful Java Web Services Security

The inherent advantages of the use of web services in computer systems development are the same that create the need for security management over them. Today, we can say that no company is able to work in complete isolation, without the need to interact with others and share and consume information. Furthermore, this is the most important asset of any company. For this reason, these requirements are also common between lines of code. This book presents real scenarios with applicable solutions, leading you by the hand all the way, so you can easily learn solutions and implementations that will resolve the most common needs that can arise.

RESTful web services offer several advantages over those based on SOAP. For example, when handling data types, depending on the programming language or the libraries you use to create them, you can find inconsistencies when using empty values ("") instead of NULL. Also, you may find difficulties in mapping complex objects and compatibility issues in file transferring when using different versions of libraries to create/consume the web service. In certain situations, even when consuming a web service created in Java from a .NET application, it ends up creating a service implemented in Java in the middle of both. This does not occur in RESTful web services, since in this case, the functionality is exposed through HTTP method invocations.

In order to protect information, the world of securities has many features that help to achieve this. For example, understanding how some issues such as authentication and authorization assist in the implementation of any selected mechanism, where the main objective is to make our applications safer and secure, is essential. The selection of each of the different ways to secure applications goes along with the problem you want to resolve; for this, we show usage scenarios for each of them.

Many times, we have seen large organizations spend time and effort in creating their own implementations to handle securities rather than using the standard that has already resolved what we need. Through the knowledge that we want to share with you, we hope to avoid this process of reinventing the wheel.

For More Information:
What This Book Covers

Chapter 1, Setting Up the Environment, helps us create our first functional application, something very similar to a Hello World example, but with some more functionality and very close to the real world. The main aim of this chapter is to familiarize ourselves with the tools we are going to use.

Chapter 2, The Importance of Securing Web Services, goes through all possible models of authentication in the Java platform. For your better understanding, we will go step by step and dive deep into how we can leverage each available authentication model. We will show you how the information is exposed and how it can be intercepted by third parties, and we will play with Wireshark, which is a very good tool to explain it. Finally, in this chapter, we will review the differences between authentication and authorization. Both concepts are very important and definitely impossible to put aside in the context of securities terms.

Chapter 3, Security Management with RESTEasy, shows how RESTEasy offers mechanisms to handle security, starting from a fairly basic model (coarse-grained) to a more elaborate one (fine-grained) in which you can perform more exhaustive controls, including managing not only configuration files, but also programmatical files.

Chapter 4, RESTEasy Skeleton Key, helps us study the OAuth implementation along with the token bearer implementation and Single Sign-On. All of them are used in order to limit the way the resources are shared. As always, you will get hands-on with code and real examples. We want to show you how sharing resources and information between applications through these technologies has turned into one of the most useful and powerful techniques by allowing clients or users to use their credentials only once to access several services, limiting the access to third-party applications to your information or data, and implementing access control through the token bearer. You will learn to apply these technologies and concepts in order to build secure and flexible applications.

Chapter 5, Digital Signatures and Encryption of Messages, helps us understand the benefits of digital signatures using a simple example; you'll notice how the message's receiver can validate the identity of the sender. In addition, we will simulate when an external agent modifies data in transit and see how digital signatures can help us to detect it, in order to avoid working with corrupted data.

Finally, we will explain SMIME for body encryption and how it works, with an example that encrypts requests and responses for your better understanding.
Setting Up the Environment

We extend you a very warm welcome to the first chapter of our journey. Let's give you an idea of what you will achieve here. After reading this chapter, you will have the basic and stimulating knowledge you need to set up a development environment to work with RESTful web services. Then, you will familiarize yourself with the development of a very basic project related to it. In addition, by the end, you will have a very clear idea of how to create applications using RESTful web services and how you can achieve this. This chapter will give you the information you need to work with web services of this kind in a very easy and comprehensive way.

In this chapter, we will cover the following topics:

- Installing the development environment
- Creating our first RESTful web services application
- Testing the RESTful web service

Downloading tools

First, we must obtain our work tools so that we get our hands into code. Tools specified here are used around the world, but you are free to choose your tools. Remember, "Tools do not make the artist". It doesn't matter if you use Windows, MAC OS X, or Linux; tools are available for every OS.

Let's explain briefly what each tool is for. We will develop the examples using Eclipse as our IDE, JBoss AS 7.1.1.Final as our application server, Maven to automatize the build process, and SoapUI as a tool to test the functionality of web services that we will create. In addition, we suggest that you should install the latest version of JDK, which is JDK 1.7.x. For help, we have obtained and included some links that you need to use to get the software to implement the first example. Each link gives you more information about each tool, which can be profitable as you learn something about each one if you don't know about them already.

For More Information:
Setting Up the Environment

Downloading links
The following tools have to be downloaded:

- Eclipse IDE for Java EE Developers 4.3 (http://www.eclipse.org/downloads/)
- JBoss AS 7.1.1 Final (http://www.jboss.org/jbossas/downloads/)
- Apache Maven 3.1.1 or higher (http://maven.apache.org/download.cgi)
- SoapUI 4.6 or higher (http://www.soapui.org/)
- JDK 1.7.x (http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html)

Creating the base project
In order to make the process of building our sample project easier, we will use Maven. This wonderful software will create a base project at the blink of an eye, and our project can be easily compiled and packaged without depending on a specific IDE.

Maven uses archetypes for a specific kind of project. The archetypes are project templates that have been previously created; they allow us to create all kinds of applications from Java desktop applications to multimodule projects, where the EAR can contain several artifacts such as JAR and WAR. Its main objective is to get users up and running as quickly as possible by providing a sample project that demonstrates many of the features of Maven. If you want to learn more about Maven, you can find more information by visiting http://maven.apache.org/.

However, the information we described here is enough to keep moving on. We will use an archetype in order to create a basic project; if we want to be more specific, we will use an archetype to create a web application with Java. To do this, we will type the following command line in a terminal:

mvn archetype:generate

When we execute this command line in a terminal, we will obtain all available archetypes in Maven's repository. So, let's look for the archetype we need in order to create our web application; its name is webapp-javaee6, and it belongs to the group org.codehaus.mojo.archetypes. Also, we can search through it using a number that represents its ID; this number is 557, as shown in the following screenshot. We recommend that you search by the name as the numbers are likely to change because some other archetypes may be added later:

For More Information:
Several questions will appear; we must provide the respective information for each question. Maven will use this information to create the archetype we selected before, as shown in the following screenshot:

```
Choose org.codehaus.mojo.archetypes:webapp-javaee6 version:
1: 1.0
2: 1.0.1
3: 1.0.2
4: 1.1
5: 1.2
6: 1.3
7: 1.4
8: 1.5
Choose a number: 8: 8
Define value for property 'groupId': com.packtpub
Define value for property 'artifactId': resteasy-examples
Define value for property 'version': 1.0-SNAPSHOT:
Define value for property 'package': com.packtpub:
Confirm properties configuration:
groupId: com.packtpub
artifactId: resteasy-examples
version: 1.0-SNAPSHOT
package: com.packtpub
```

As you have probably noticed, each question asks you to define a property, and each property is explained as follows:

- **groupId**: This property represents the company's domain reversed order; this way we can recognize which company is the code's owner
- **artifactId**: This property represents the project's name
- **version**: This property represents the project's version
- **package**: This property represents the base package's name where classes are going to be added

For More Information:

Class names and package names together shape the class's full name. This full name allows the class names to be identified in a unique way. Sometimes, when there are several classes with the same name, the package name helps to identify which library it belongs to.

The next step is to put the project into Eclipse's workspace; to do this, we must import our project into Eclipse by navigating through File | Import | Maven | Existing Maven Projects.

We should see the project in the IDE, as shown in the following screenshot:

![Screenshot of Eclipse IDE with project displayed]

Before moving on, let's fix the problems that have occurred in the file `pom.xml`.

The error shown in the following code is related to a bug that comes from Eclipse and Maven integration. In order to fix this, we have to add the `<pluginManagement>` tag after the `<build>` tag.
The pom.xml file should look like the following:

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.packtpub</groupId>
  <artifactId>resteasy-examples</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  ...
  <build>
    <pluginManagement>
      <plugins>
        <plugin>
          ...
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

Downloading the sample code

You can download the sample code files for all Packt books you have purchased from your account at http://www.packtpub.com. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you. Also, we highly suggest obtaining the source code from GitHub available at https://github.com/restful-java-web-services-security.

For More Information:
This will fix the error, and now we only need to update Maven's configuration in the project, as shown in the following screenshot:

After refreshing the project, the errors should go away because when we update Maven's configuration we are actually updating our project's dependencies, such as missing libraries. Through this, we will include them in our project and errors will disappear.

Inside the src/main/webapp path, let's create the WEB-INF folder.

Now, inside the WEB-INF folder, we will create a new file named web.xml with the following content:

```xml
<?xml version="1.0" encoding="UTF-8"?>
</web-app>
```

This file is very useful when you are securing your applications; this time, we will create it without any configuration. For now, the WEB-INF folder and the web.xml file only define the structure of the web application.

---

For More Information:
First functional example

Now that we have our development environment all set up, it is time to get your hands dirty and write the first RESTful web service. As we are using JBoss, let's use the RESTEasy implementation for JAX-RS. We will develop a very simple example; let's imagine you want to implement a service to save and search for people's information.

First, we create a simple Person domain class that uses JAXB annotations. JAXB marshals/unmarshals objects between XML and Java. For this example, we'll store these instances in an in-memory cache instead of a database. In JEE, this typically represents a table in a relational database, and each entity instance corresponds to a row in that table, as presented in the following code:

```java
package com.packtpub.resteasy.entities;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "person")
@XmlAccessorType(XmlAccessType.FIELD)
public class Person {

    @XmlAttribute
    protected int id;

    @XmlElement
    protected String name;

    @XmlElement
    protected String lastname;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {

```
Setting Up the Environment

```java
return name;
}

public void setName(String name) {
    this.name = name;
}

public String getLastName() {
    return lastname;
}

public void setLastName(String lastname) {
    this.lastname = lastname;
}

Next, we create a new class called PersonService in the com.packtpub.resteasy.services package. This class will have two methods; one to register a new person and another to search for people by ID. This class will store people using an in-memory map cache.

The service will have the following implementation:

```java
package com.packtpub.resteasy.services;

import java.net.URI;
import java.util.HashMap;
import java.util.Map;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Response;
import com.packtpub.resteasy.entities.Person;

@Path("/person")
public class PersonService {
    private Map<Integer, Person> dataInMemory;
    public PersonService() {
```
Chapter 1

dataInMemory = new HashMap<Integer, Person>();

@POST
@Consumes("application/xml")
public Response savePerson(Person person) {
    int id = dataInMemory.size() + 1;
    person.setId(id);
    dataInMemory.put(id, person);
    return Response.created(URI.create("/person/" + id)).build();
}

@GET
@Path("{id}")
@Produces("application/xml")
public Person findById(@PathParam("id") int id) {
    Person person = dataInMemory.get(id);
    if (person == null) {
        throw new WebApplicationException(Response.Status.NOT_FOUND);
    }
    return person;
}

The @Path annotation defines the path in the URL that will be available on the functionalities that have been written within this class. The method annotated with @Post indicates that it should make a HTTP POST request. Furthermore, it is annotated with @Consumes and uses the application/xml value; this means that the POST request will be performed with a string in XML format, containing the information of the person to be saved. On the other hand, to find a person from its ID, you must make an HTTP GET request. The URL must indicate the ID the same way as indicated by the @Path annotation on the method. The @Produces annotation indicates that we will get the response in XML format. Finally, notice that the parameter ID, as indicated in the @Path annotation, is used as an argument of the method using the @PathParam annotation.

Finally, we write a class that will extend the Application class and set the service we just created as a singleton. So, the information won’t get lost in every request, and we will keep it in memory as follows:

package com.packtpub.resteasy.services;

import java.util.HashSet;

For More Information:
Setting Up the Environment

```java
import java.util.Set;
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/services")
public class MyRestEasyApplication extends Application {

    private Set<Object> services;

    public MyRestEasyApplication() {
        services = new HashSet<Object>();
        services.add(new PersonService());
    }

    @Override
    public Set<Object> getSingletons() {
        return services;
    }
}
```

Note that as we have mapped our entity using JAXB, our methods consume and produce information in the XML format.

In order to deploy our application in JBoss, we should add a dependency in the `pom.xml` file. This dependency must reference to the JBoss plugin. We have to change the generated artifact name in `pom.xml`. The default value for this is the `artifactId` file, followed by the version; for example, `resteasy-examples-1.0-snapshot.war`. We will set it, so we will use just the `artifactId` file; in this case, `resteasy-examples.war`. All of these configurations must be included, modified, and implemented in `pom.xml`, as shown in the following piece of XML code:

```xml
<build>
    <finalName>${artifactId}</finalName>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.jboss.as.plugins</groupId>
                <artifactId>jboss-as-maven-plugin</artifactId>
                <version>7.5.Final</version>
                <configuration>
                    <jbossHome>/path/to/jboss/jboss-as-7.1.1.Final</jbossHome>
                </configuration>
            </plugin>
        </plugins>
    </pluginManagement>
</build>
```

For More Information:

You should change the value of the \texttt{jbossHome} property for the path of your JBoss installation. After this, we will use the command terminal; head to the project's directory, and type\texttt{mvn jboss-as:run}. If you make any change on the code after the command has been executed, then you should use the following command in order to see the changes:

\texttt{mvn jboss-as:redeploy}

Run and redeploy are the goals of this plugin. If you want to know more goals about this plugin, you can visit \url{https://docs.jboss.org/jbossas/7/plugins/maven/}

This will compile all project classes again; it will then be packaged in order to create the .war file. At the end, the modifications will be deployed on the server. If everything is okay, we should see a message in the terminal saying that the deployment has been done successfully, as shown in the following screenshot:

The source code of this chapter is available on GitHub at the following location:

\url{https://github.com/restful-java-web-services-security/source-code/tree/master/chapter01

Setting Up the Environment

Testing the example web service

At this moment, we will test the functionality we just created. We will use SoapUI as our test tool; make sure you use the latest version, or at least the version equal to or greater than 4.6.x because this version offers more features to test the RESTful Web services. Let's start by performing the following steps:

1. From the main menu, let's create a new REST project by navigating to **File | New REST Project**, as shown in the following screenshot:

![SoapUI New REST Project](image)

2. Set the URI of our service, as follows:

![New REST Project URI](image)

3. After this, let's create a new person using the **POST** method from workspace. In the field **Media Type**, select **application/xml** and perform a request with a string that contains the XML with the information, as shown in the following text:

```xml
<person>
  <name>Rene</name>
  <lastname>Enriquez</lastname>
</person>
```

For More Information:

4. When we click on the **Play** button, we should obtain an answer where it shows the created resource URI (hyperlink "http://localhost:8080/resteasy-examples/services/person/1"), as shown in the following screenshot:

![Screenshot](image)

5. If we change the URI from the **Resource** textbox in SoapUI and use the **GET** method, it will show us the data we just entered, as shown in the following screenshot:

![Screenshot](image)

Congratulations! We have developed our first functional RESTful web service with two features. The first is to keep people's information in memory, and the second is to retrieve people's information through an ID.

If you restart JBoss or deploy the application again, all data will be lost. Before searching for people's information, you must first save the data.

---

For More Information:

Summary

In this chapter, we created our first functional application—something like a *hello world* example but with a bit more functionality close to the real world.

The essential part we covered in this chapter is to familiarize ourselves with the tools we will use. In later chapters, we will assume that these concepts are already clear. For example, we will move forward step-by-step when using SoapUI as this is a tool that will facilitate the task of testing the functionality that we will be developing. This way, we will avoid the task of writing code for web service clients.

Now we are ready to review the next chapter, which contains some security models that Java provides. We will understand each one of them and learn how to implement them.

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

For More Information: