Chat Application

The application works like a mini-BBS. Users login to the application with a nickname. The user session is saved in a session scoped component. Once logged in, they can leave quips or messages.

Action Classes

`LoginAction` extends `ActionSupport`, which has default implementations of some of the basic action methods and provides some validation support.

The two main methods we are concerned with in `ActionSupport` are `validate()` and `execute()`.

Validation

Validation is performed on your Action class if it implements `Validatable` (`ActionSupport` does), and your DefaultWorkflowInterceptor is activated on that action. The default configuration includes the workflow interceptor.

Execution

`execute()` returns a String. This String will be used to determine which result is used.

The framework provides some default return Strings, namely:

```
Action.SUCCESS = "success"
Action.INPUT = "input"
Action.NONE = "none"
Action.ERROR = "error"
Action.LOGIN = "login"
```

For example, lets take a look at the relevant part of our `struts.xml` configuration for LoginAction...

struts.xml fragment

```
<action name="login" class="example.LoginAction">
  <result name="success" type="chain">
    <param name="actionName">viewMessages</param>
  </result>

  <result name="input" type="chain">
    <param name="actionName">viewMessages</param>
  </result>
</action>
```

If `execute()` returns a String of "success", the result with attribute "success" will be used. If `execute()` returns a String of "input", the result with attribute "input" will be used.

You can define your own return results. For example:

```
public String execute() {
  return "resetPassword";
}
```

We need only configure a result for the "resetPassword" name:

```
<action name="login" class="example.LoginAction">
  <result name="resetPassword" type="chain">
    <param name="actionName">viewResetPassword</param>
  </result>
</action>
```

Context Variables / Mapping

LoginAction.java

```
public class LoginAction extends ActionSupport {

    private String loginName;

    public String getLoginName() {
        return loginName;
    }

    public void setLoginName(String loginName) {
        this.loginName = loginName;
    }
}
```

`LoginAction` has a bean property `loginName`. This property will be set automatically by Struts 2 from forms containing a `loginName` element.

```
<form method="POST" action="login.action">
  <input type="text" name="loginName" size="20">
  <input type="submit" value="Login">
</form>
```

Also, the bean property is available to your views. In Velocity, this accessible via the VelocityContext.

```
$loginName
```

which is mapped to `getLoginName()` in our `LoginAction` class.

You can map any other object you wish. For example, I could have a `User` object:

```
class User {
   private Account account;
   private String name;
   // with the relevant getX() methods...
}
```

We add a `user` property to an action class:

```
class MyExampleAction {
   //...
   User getUser() {
      returns user;
   }
   //...
}
```

In our Velocity template we can access `User` properties as expected:

```
Welcome, $user.name.
You last logged on at $user.lastLogin.
You currently have $user.account.balance left in your account.
```

Components

components.xml

```
<components>
    <component>
        <scope>application</scope>
        <class>example.data.MessageList</class>
        <enabler>example.data.MessageListAware</enabler>
    </component>

    <component>
        <scope>session</scope>
        <class>example.web.UserSession</class>
        <enabler>example.web.UserSessionAware</enabler>
    </component>
</components>
```

Two components, one to hold the application scoped chat messages, another to hold the user's session information (login account name, etc.) For all practical purposes, you can replace the application scoped component with a database. i.e. instead of reading/writing to the component, read/write to the database.