```
package br.com.optimedia.autor.model
{
	import br.com.optimedia.assets.FaultHandler;
	import br.com.optimedia.assets.NotificationConstants;
	import br.com.optimedia.assets.vo.CompleteUserVO;
	import br.com.optimedia.assets.vo.PresentationVO;
	import br.com.optimedia.assets.vo.SubjectVO;
	import br.com.optimedia.autor.AutorFacade;

	import mx.collections.ArrayCollection;
	import mx.controls.Alert;
	import mx.rpc.AsyncToken;
	import mx.rpc.Responder;
	import mx.rpc.events.FaultEvent;
	import mx.rpc.events.ResultEvent;
	import mx.rpc.remoting.mxml.RemoteObject;

	import org.puremvc.as3.multicore.patterns.proxy.Proxy;

	public class SubjectManagerProxy extends Proxy
	{
		public static const NAME:String = "SubjectManagerProxy";

		private var remoteService:RemoteObject;

		public function SubjectManagerProxy(data:Object=null)
		{
			super(NAME, data);
		}

		override public function onRegister():void {
			trace(NAME+".onRegister()");
			remoteService = new RemoteObject();
			remoteService.destination = "amfphp";
			remoteService.source = "autor.SubjectManager";
			remoteService.showBusyCursor = false;
			getSkins();
		}

		private function generalFault(event:FaultEvent):void {
			FaultHandler.handleFault(event);
		}

		public function getSubjects():void {
			var userID:int;
			//if( AutorFacade(facade).userRole ==
AutorFacade.IS_EDITOR ) {
				userID = AutorFacade(facade).userID;
			//}
			var asynkToken:AsyncToken =
remoteService.getSubjects(userID);
			asynkToken.addResponder( new
Responder(getSubjectsResult, generalFault) );
		}
		private function getSubjectsResult(event:ResultEvent):void {
			sendNotification( NotificationConstants.GET_SUBJECTS_OK,
event.result );
		}
```

```
            public function saveSubject(subjectVO:SubjectVO):void {
                  var asynkToken:AsyncToken =
remoteService.saveSubject(subjectVO);
                  asynkToken.addResponder( new
Responder(saveSubjectResult, generalFault) );
            }
            private function saveSubjectResult(event:ResultEvent):void {
                  if( event.result == true ) {
                        sendNotification(
NotificationConstants.SAVE_SUBJECT_OK, event.result );
                        getSubjects();
                  }
                  else Alert.show("Não foi possível salvar, verifique os
campos e tente novamente", "Erro");
            }

            public function
savePresentation(presentationVO:PresentationVO):void {
                  var asynkToken:AsyncToken =
remoteService.savePresentation(presentationVO);
                  asynkToken.addResponder( new
Responder(savePresentationResult, generalFault) );
            }
            private function
savePresentationResult(event:ResultEvent):void {
                  if( event.result == true ) {
                        sendNotification(
NotificationConstants.SAVE_PRESENTATION_OK, event.result );
                        getSubjects();
                  }
                  else if( event.result is String ) {
                        Alert.show("Esta apresentação está bloqueada por
"+event.result, "Erro");
                  }
                  else Alert.show("Não foi possível salvar, verifique os
campos e tente novamente", "Erro");
            }

            [Bindable]
            public var presentationSkins:ArrayCollection = new
ArrayCollection();

            public function getSkins():void {
                  var asynkToken:AsyncToken = remoteService.getSkins();
                  asynkToken.addResponder( new Responder(getSkinsResult,
generalFault) );
            }
            private function getSkinsResult(event:ResultEvent):void {
                  if(event.result is Array) {
                        presentationSkins = new
ArrayCollection(event.result as Array);
                        sendNotification(
NotificationConstants.GET_SKINS_RESULT, event.result );
                  }
            }

            public function deleteSubject(subjectVO:SubjectVO):void {
                  var asynkToken:AsyncToken =
remoteService.deleteSubject(subjectVO.subject_id);
                  asynkToken.addResponder( new
Responder(deleteSubjectResult, generalFault) );
            }
            private function deleteSubjectResult(event:ResultEvent):void {
                  if( event.result == true ) {
                        sendNotification(
NotificationConstants.DELETE_SUBJECT_OK );
                        getSubjects();
                  }
                  else Alert.show("Não foi possível excluir, verifique se
este módulo não possui temas", "Erro");
            }

            public function
deletePresentation(presentationVO:PresentationVO):void {
                  var asynkToken:AsyncToken =
remoteService.deletePresentation(presentationVO.presentation_id);
                  asynkToken.addResponder( new
Responder(deletePresentationResult, generalFault) );
            }
            private function
deletePresentationResult(event:ResultEvent):void {
                  if( event.result == true ) {
                        sendNotification(
NotificationConstants.DELETE_PRESENTATION_OK );
                        getSubjects();
                  }
                  else Alert.show("Não foi possível excluir.", "Erro");
            }

            public function
publishPresentation(presentationVO:PresentationVO, sectionID:uint):void {
                  var asynkToken:AsyncToken =
remoteService.publishPresentation(presentationVO.presentation_id,
sectionID, presentationVO.title);
                  asynkToken.addResponder( new
Responder(publishPresentationResult, generalFault) );
            }
            private function
publishPresentationResult(event:ResultEvent):void {
                  if( event.result == true ) {
                        sendNotification(
NotificationConstants.PUBLISH_PRESENTATION_OK );
                        getSubjects();
                  }
                  else Alert.show("Não foi possível publicar.", "Erro");
            }

            public function unpublishPresentation(presentationID:uint,
sectionID:uint):void {
                  var asynkToken:AsyncToken =
remoteService.unpublishPresentation(presentationID, sectionID);
                  asynkToken.addResponder( new
Responder(unpublishPresentationResult, generalFault) );
            }
```

```
		private function
unpublishPresentationResult(event:ResultEvent):void {
			if( event.result == true ) {
				sendNotification(
NotificationConstants.UNPUBLISH_PRESENTATION_OK );
				getSubjects();
			}
			else Alert.show("Não foi possível despublicar.",
"Erro");
		}

		public function getSections():void {
			var asynkToken:AsyncToken = remoteService.getSections();
			asynkToken.addResponder( new
Responder(getSectionsResult, generalFault) );
		}
		private function getSectionsResult(event:ResultEvent):void {
			if( event.result is Array ) {
				sendNotification(
NotificationConstants.GET_SECTIONS_RESULT, event.result );
			}
			else Alert.show("Não foi possível recuperar nenhum
setor.", "Erro");
		}

		public function lockPresentation(presentationID:uint,
userID:uint):void {
			var asynkToken:AsyncToken =
remoteService.lockPresentation(presentationID, userID);
			asynkToken.addResponder( new
Responder(lockPresentationResult, generalFault) );
		}
		private function
lockPresentationResult(event:ResultEvent):void {
			if( event.result == true ) {
				sendNotification(
NotificationConstants.LOCK_PRESENTATION_OK );
				sendNotification(
NotificationConstants.ENABLE_SLIDE_EDITION);
			}
			else if( event.result is CompleteUserVO ) {
				Alert.show("Esta apresentação está bloqueada por
"+CompleteUserVO(event.result).first_name+"
"+CompleteUserVO(event.result).last_name+". Você não poderá editar os
Slides.", "Erro");
				sendNotification(
NotificationConstants.DISABLE_SLIDE_EDITION,
CompleteUserVO(event.result).user_id );
			}
			else if( event.result == false ) {
				Alert.show("Erro ao bloquear apresentação",
"Erro");
				sendNotification(
NotificationConstants.DISABLE_SLIDE_EDITION,
CompleteUserVO(event.result).user_id );
			}
		}
```

```
            public function unlockPresentation(presentationID:uint):void {
                  var asynkToken:AsyncToken =
remoteService.unlockPresentation(presentationID);
                  asynkToken.addResponder( new
Responder(unlockPresentationResult, generalFault) );
            }
            private function
unlockPresentationResult(event:ResultEvent):void {
                  if( event.result == true ) {
                        sendNotification(
NotificationConstants.UNLOCK_PRESENTATION_OK );
                  }
                  else Alert.show("Não foi possível destravar a
apresentação.", "Erro");
            }
      }
}
```