Savant Web Framework

by Hendrik Heinemann

by Hendrik Heinemann

Table of Contents

Draft area for "book" metainfo.

Savant Web Framework

personname: Hendrik Heinemann`<hendrik.heinemann@googlemail.com>`Copyright © 2010° Hendrik Heinemann

Chapter 1. Installation

Draft area for "chapter" metainfo.

Installation

Section 1.1. Requirements

Draft area for "section" metainfo.

Requirements

The following minimum requirement should be operating system independent. Savant has been tested on Ubuntu Linux 10.04 (Debian fork) with an Apache webserver but Windows systems should should work as well.

- PHP 5.3 or newer (but PHP6 is not supported yet)
  - Configuration: gpc_magic_quotes = off
  - Modules: pdo_sqlite
  - PECL Modules: bcompiler, PHPUnit
- Apache
  - Modules: mod_rewrite

Section 1.2. Installation of Savant

Draft area for "section" metainfo.

Installation of Savant

The following steps require the server requirements to be set up.

1. Unzip it to a folder accessible by the webserver
2. Change the document root in the webserver configuration to `htdocs` folder in the Savant folderstructure
3. The webserver needs write permissions for some subfolders

To check if Savant works correctly, open

uri:

http://*<hostname/>*/info.php

. If everything works correctly you should see the phpinfo-screen. To use beautified URLs mod_rewrite has to be activated and the configuration `.htaccess` file has to be used.

Chapter 2. AOP Framework

Draft area for "chapter" metainfo.

AOP Framework

Section 2.1. Introduction

Draft area for "section" metainfo.

Introduction

Aspect-Oriented Programming(AOP) is a programming paradigm which extends Object-Oriented Programming(OOP) to improve the applications modularization. The separation of concerns aims for making an application easier to maintain by grouping behaviour into manageable parts which have a specific usecase to take care of.

Some concerns a difficult to handle as they cross the boundaries of several classes or packages. Many classes of an application need a similar functionality like for example Configuration. This is called a cross-cutting concern. AOP enables you to move the configuration aspect into its own package and leave other objects with clear responsibilities.

Savant does not provide full AOP support, just the parts which are needed by the framework itself are implemented yet.

Section 2.2. Terminology

Draft area for "section" metainfo.

Terminology

This chapter explains some aop terms and introduces a bit more of the concept.

Aspect
: An aspect is an cross-cutting concern which takes care of functionality that is subject of multiple objects in the framework. Savant handles aspects as regular classes which must implement the `Savant\AOP\IAspect` and can derive from `Savant\AOP\AAspect` to use default aspect behaviour. The methods of an aspect class represent advices.

Joinpoint
: A joinpoint is an entry-point in the flow of the program, for example the execution of a method, loading a class or throwing an exception. A joinpoint defines an event which occurs during the programflow. In Savant joinpoints must derive from `Savant\AOP\AJoinPoint` and should be declared in the `Savant\AOP\JoinPoints` namespace.

Advice
: An advice is the action taken by an aspect at a particular joinpoint. It has to be implemented as a method of the aspect class and will be executed before or after the joinpoint is reached. The joinpoint and the executed object has to be passed as parameters to the advice.

Pointcut
: A pointcut defines a set of joinpoints which needed to be matched before running an advice.

Section 2.3. Aspects

Draft area for "section" metainfo.

Aspects

advices bla bla

Example 2.3.1. Declaration of an aspect

```
namespace Savant\AOP\Aspects;
use Savant\AOP;

abstract class MyAspect extends AOP\AAspect implements
AOP\IAspect
{
}
```

Section 2.4. Pointcuts

Draft area for "section" metainfo.

Pointcuts

pointcuts bla bla

Example 2.4.2. Declaration of a pointcut

programlisting

Chapter 3. Webservices

Draft area for "chapter" metainfo.

Webservices

This chapter introduces Savant's webservices to the developer. The framework supports differend kinds of webservices.

Section 3.1. SOAP Webservices

Draft area for "section" metainfo.

SOAP Webservices

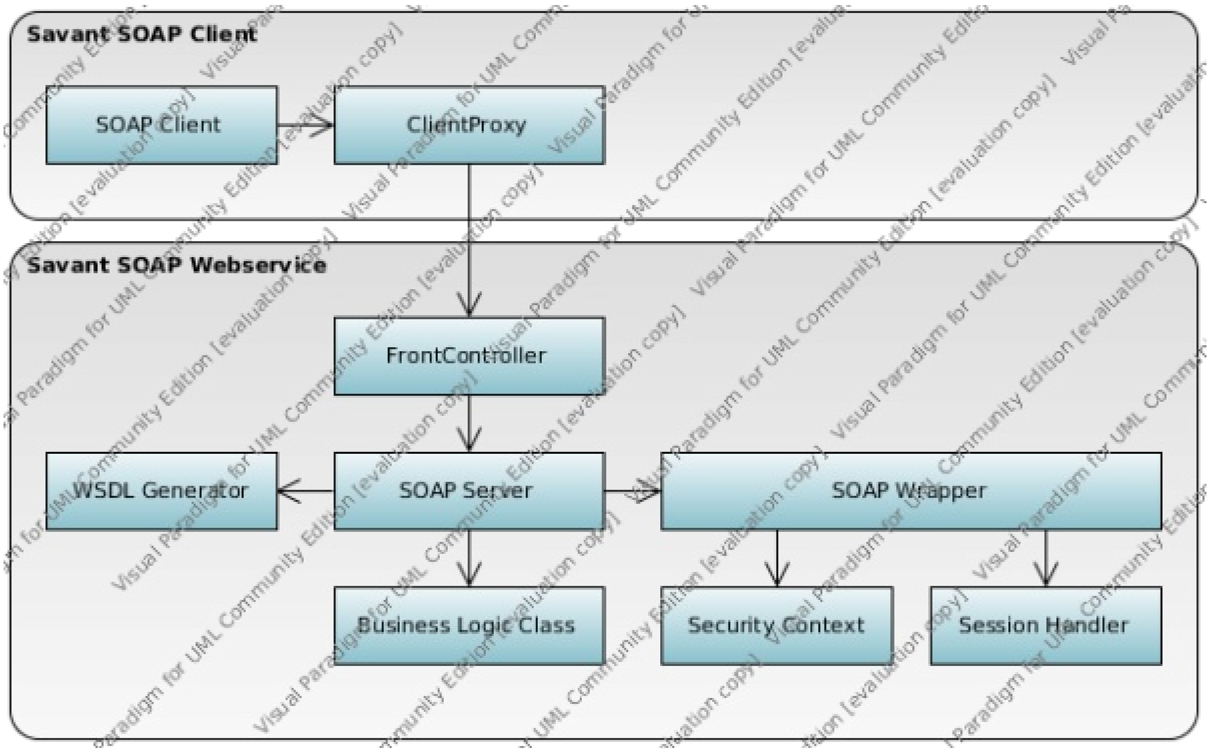
This section describes SOAP Webservices and the way they are handled by Savant.

Section 3.1.1. Architecture

Draft area for "section" metainfo.

Architecture

Figure 3.1.1.1. Introduction to Webservice Framework



The webservice architecture in Savant hides the complexity of SOAP Webservices from the developer. Annoying tasks like writing wsdl files are no longer necessary. The framework generates proxy objects called “stubs” automatically around both client and server objects.

Section 3.2. Document/Literal Wrapped Style with Session Handling

Draft area for "section" metainfo.

Document/Literal Wrapped Style with Session Handling

The `Savant\Webservice\CSoapServer` class can be used to promote any business logic class with deterministic parameters to a SOAP webservice with automated security handling.

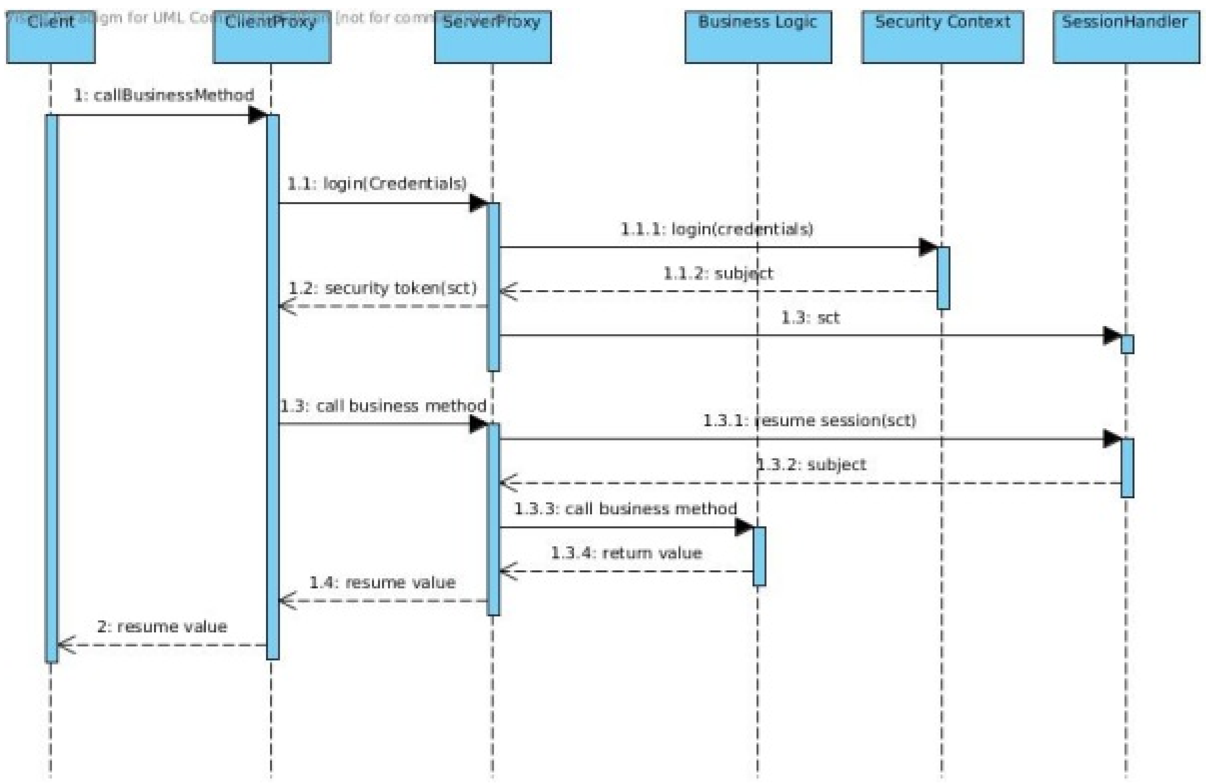
Section 3.2.1. Communication Sequence

Draft area for "section" metainfo.

Communication Sequence

The following communication sequence describes the communication between the client of a Savant Secure Soap Webservice and the server

Figure 3.2.1.2. Subsequent call


Client
ClientProxy
ServerProxy
Business Logic
Security Context
SessionHandler
1: callBusinessMethod
1.1: login(Credentials)
1.1.1: login(credentials)
1.1.2: subject
1.2: security token(sct)
1.3: sct
1.3: call business method
1.3.1: resume session(sct)
1.3.2: subject
1.3.3: call business method
1.3.4: return value
1.4: resume value
2: resume value