DocNo: 001.E.1.1

**Grape**

**Software Architecture**

**Document**

**Version 1.0**

**By**:

Group Undefined

2015-04

**Group Member**:

Hunter Lin

Birdy

Listen

Morning

Syachi

**Document Language**:

English

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 2015.4.12 | 1.0 | Initialization of the report | Hunter Lin |
| Final Date | 2.0 |  |  |

Contents

[**1.** **Introduction** 5](#_Toc417418435)

[1.1. Purpose 5](#_Toc417418436)

[1.2. Scope 5](#_Toc417418437)

[1.3. Reference 5](#_Toc417418438)

[**2.** **Architectural Representation** 5](#_Toc417418439)

[**3.** **Architectural Goals and Constraints** 5](#_Toc417418440)

[**4.** **Use-Case View** 5](#_Toc417418441)

[4.1 Overview 5](#_Toc417418442)

[4.2 Architecturally Significant use cases 5](#_Toc417418443)

[**5.** **Logical View** 5](#_Toc417418444)

[5.1. Overview 5](#_Toc417418445)

[5.2. Front-end Interaction Mechanisms 6](#_Toc417418446)

[5.2.1. Front Controller 6](#_Toc417418447)

[5.2.2. Command Delegator 6](#_Toc417418448)

[5.2.3. Service Locator 6](#_Toc417418449)

[5.2.4. Security Handler 6](#_Toc417418450)

[5.3. Data Operation Mechanisms 6](#_Toc417418451)

[5.3.1. Persistency 6](#_Toc417418452)

[5.3.2. Session Facade 6](#_Toc417418453)

[5.4. Architecturally Significant Use Case Realization 6](#_Toc417418454)

[5.5. Architecturally Significant Model Elements 6](#_Toc417418455)

[5.6. Architecturally Significant Classes 7](#_Toc417418456)

[**6.** **Process View** 7](#_Toc417418457)

[**7.** **Deployment View** 7](#_Toc417418458)

[**8.** **Implementation View** 7](#_Toc417418459)

[**9.** **Size and Performance** 7](#_Toc417418460)

[**10.** **System Size** 7](#_Toc417418461)

**蓝色部分：Syachi**

**黄色部分：Hunter Lin**

**红色部分：Listen**

**紫色部分：Morning**

**绿色部分：Birdy**

1. **Introduction**

## 1.1. Purpose

## 1.2. Scope

## 1.3. Reference

1. **Architectural Representation**

First of all, let’s just give a general overview on the representations of the architectural layers. The following document sections will be constructed as illustrated below.

This document presents the architectural as a series of views:

1. Use Case View
2. Logical View
3. Process View
4. Implement View
5. Deploy View

Each of the view above is just a different prospect of looking at our system in order to get a clearer concept. The architecture of our Grape system is represented by the recommended software “PowerDesigner”, which will give us a instant simple graph.

Note that the Logical View and the Component View also include packages that represent html front & end language (plus the models we multiplex) and python framework elements. Collectively the above models and packages form a complete UML specification of the system.

1. **Architectural Goals and Constraints**

The architectural goal of this document is to give the programmer several prospect of views to look at our system, thus grasping some deeper concepts in the real programming level.

There are some key requirements (goals for developing) that have a significant bearing on the architecture. We will list it below:

1. Provide an on-line interactive platform for different users to communicate and share their resources and opinions.
2. Allow group leaders to track and analyze the effect of their activity.
3. Allow group members to establish a closer relationship with other group members and the leader.

There are some key system constraints that have a significant bearing on the architecture. We will list it below:

1. The system must ensure complete protection of data from unauthorized access. All accesses are subject to user identification and password control. For example, the user who does not belong to a group should have no access to that group’s detail.
2. The system will be implemented as a web browser-server system. The users access the system from Internet explorer in their on PCs and the server portion must operate on the Windows Server in the company intranet.
3. All performance and loading requirements, as stipulated in the Vision Document and the Supplementary Specification, must be taken into consideration as the architecture is being developed.
4. The system should allow at least 10000 people browsing and operating on the website simultaneously. That is to say, the throughput of our system must be large enough.
5. **Use-Case View**

## 4.1 Overview

## 4.2 Architecturally Significant use cases

1. **Logical View**

This section depicts firstly some important mechanisms in design model, most of which are generated by Design Patterns. Secondly, we describe the architecturally significant parts of the design model, such as the decomposition into subsystems and packages, and the logical structure of our system. We’ll start from the overview of the architecture, giving a direct and general view of the contents, then the presentation of the important structure, behavioral elements and other evaluations.

## 5.1. Overview

There are three dominant structures in the application design model:

1. Logical decomposition of the system into three layers.
2. The structure of the use case realizations derived from design patterns. Note that these mechanisms include some of the pre-defined solutions to facilitate our further implementations.

The high-level diagram of above is showed below:

You can see many mechanisms in the design model. Some of these mechanisms are derived from design patterns. In fact, the mechanisms we depict here can be of great use to any developer who intends to create a system with group operations. We use mechanisms to provide pre-designed solutions to some common problems that have to be addressed repeatedly in the application and to unify the designs of every part. That would significantly reduce our workloads.

In our grape system, two kinds of mechanisms exist:

1. Front-end Interaction with Other Components:
   1. Front Controller
   2. Command Delegator
   3. Service Locator
   4. Security Handler
2. Data Access and Operation
   1. Persistency
   2. Session Façade

These two kinds of mechanisms will be addressed below. In the following two sections, we will organize each mechanism in a strict and clear order. First, a class diagram and a sequence diagram will be displayed. Then, we will introduce how the mechanism works and the situation we apply it to our Grape system. Finally, we will address the reason why we choose this mechanism, and the advantages of using this mechanism.

## 5.2. Front-end Interaction Mechanisms

### 5.2.1. Front Controller

Class diagram and sequence diagram:

### 5.2.2. Command Delegator

### 5.2.3. Service Locator

### 5.2.4. Security Handler

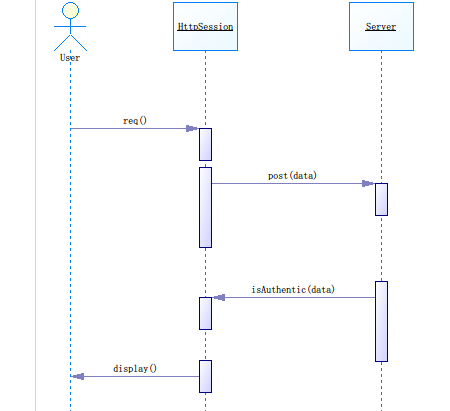
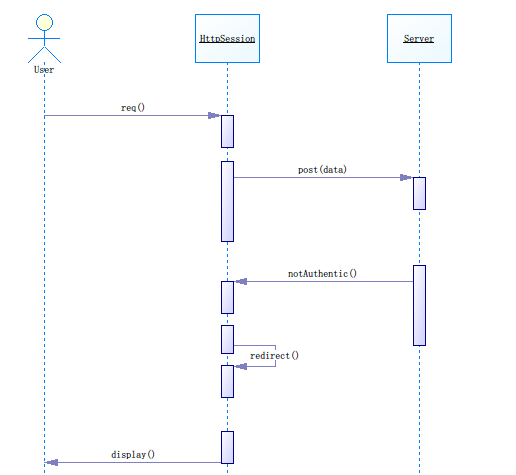
**How it works:**

When the user wants to use the system,the system will check whether he has logged in.If not,it will redirect to the page for logging in.And if the user has logged in,the system will identify his role(admin or normal user).Then corresponding function will be displayed.

**Why we use it:**

To avoid the users access the resources, which they have no authority to access.

### sequence diagram:



## 5.3. Data Operation Mechanisms

### 5.3.1. Persistency

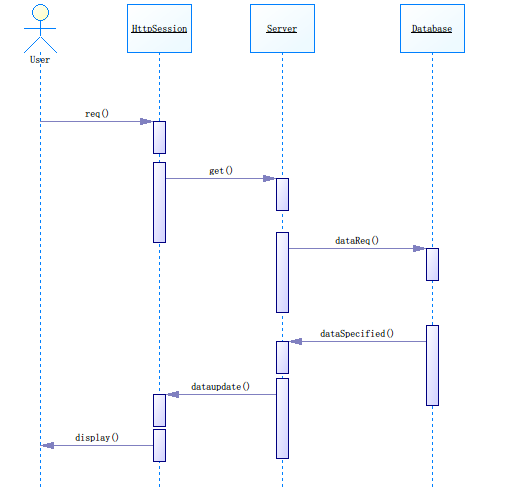
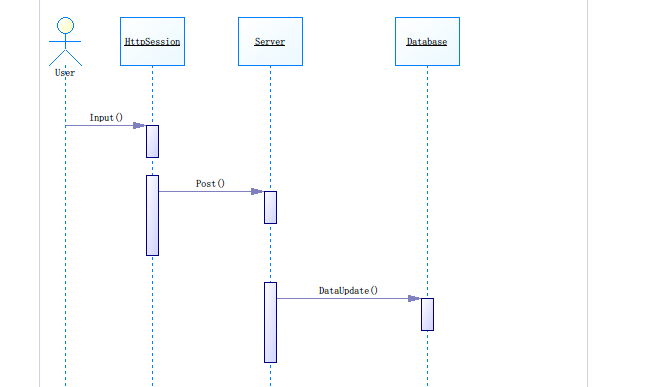
**How it works:**

In our system,we use mysqldb for python to connect MySQL database.One of the convenience is that you we can use the same SQL instruction in python by mysqldb.And it’s therefore simple to operate dynamic change of all kinds of data .

**Why we use it:**

It’s essential to keep the data permanently for further use.Obviously it’s a basic function of all websites.

### sequence diagram:



### 5.3.2. Session Facade

This Pattern is not addressed in the Design Mechanism package in the Design Model. But every one in our team must use this pattern, when they are designing their own use cases.

**How it works:**

The mechanism is a “blueprint” for the organization of access to the application server components. The architecture does not allow any presentation layer components to communicate directly with entity EJBs. Hence, the only beans that can be accessed remotely are session EJBs and the mechanism is used for that.

**Key Point:**

1. All business service components are implemented as Session EJBs or have Session EJB façades.
2. There is always no one-to-one relationship between SessionFacade and the Entity beans. In this system how SessionFacade Manage its EntityBeans will be depict below as Architecturally-significant Model elements.
3. In this system, a BusinessFacade are used by a BusinessDelegator derectly.

**Advantages:**

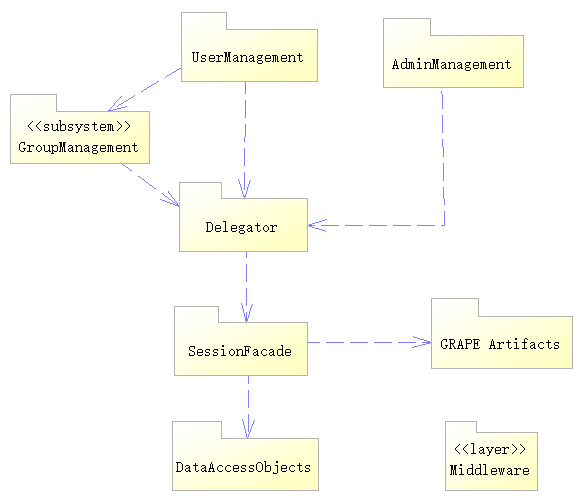
1. Session Facades can represent a control layer between clients and the business tier, as identified through analysis modeling.
2. The underlying interactions between the business components can be very complex. A Session Facade pattern abstracts this complexity and presents the client a simpler interface that is easy to understand and to use.
3. Reduces Coupling, Increases Manageability.
4. Improves Performance, Reduces Fine-Grained Methods.
5. Centralizes Transaction Control.
6. Exposes Fewer Remote Interfaces to Clients.

## 5.4. Architecturally Significant Use Case Realization

Addressed in the design model, please reference it.

## 5.5. Architecturally Significant Model Elements

**Architecturally-Significant Packages:**



**AdminManagement:** AdminManagement related classes , admin operations must be put in this package.

**UserManagement:** UserManagement related classes, user operations must be put in this package.

**GroupManagement SubSystem:** GroupManagemn related classes, group operations must be put in this package.

**Delegator:** business delegate classes

**SessionFacade:** business session façade.

**GRAPE Artifacts:** python, flask

**DataAccessObject:** Data Access Objects, see mechanism in 5.2.

## 5.6. Architecturally Significant Classes

**Front controller:** see front controller mechanism in 5.2.

**Abstract Command:** see front controller mechanism in 5.2.

**Abstract CommandFactory:** see front controller mechanism

**CommandFactoryImpl:** see front controller mechanism.in 5.2

**SecurityValidator:** see Security mechanism in 5.2.

**Errorpage:** see Secutiry mechanism in 5.2.

All of the above exist in the Application layer.

**ServiceLocator:** See serviceLocator Mechanism in 5.2.

**Subsystem interfaces:** Iauthoritiesmanager and Istaffinfosystem.

1. **Process View**
2. **Deployment View**
3. **Implementation View**
4. **Size and Performance**

The chosen software architecture supports the key sizing and timing requirements:

1. The system shall support up to 2000 simultaneous users against the central database at any given time, and up to 1000 simultaneous users against the local servers at any one time.
2. The system shall provide access to the legacy course catalog database with no more than 10-second latency.
3. The system must be able to complete 80% of uploading operations in at most 2 minutes.
4. The system must be able to complete 80% of the downloading operations in at most 1 minutes.
5. **System Size**

The Grape system’s size can be described with the following indexes:

1. Labor months: 5
2. Business components: 5
3. Dependencies on external components: 4
4. Lines of total coding: 8,000
5. Source file number: 100
6. Implemented use cases: 30

Note:

The coding language may include HTML, CSS, Javascript, Python, Ajax. So the total coding lines is the sum of the above languages.