DocNo: 001.E.1.1

Grape

Software Architecture

Document

Version 2.0

By:

Group Undefined

2015-05

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
2015.4.12	1.1	Finished the part of section1,2	Syachi
2015.4.12	1.2	Finished the part of section3,4	Listen
2015.4.12	1.3	Finished the part of section5,6	Birdy
2015.4.12	1.4	Finished the part of section7,8,9	Morning
Final Date	2.0	Integration of all the works	Hunter Lin

Contents

1. Introduction

1.1. Purpose

This document provides a comprehensive architectural overview of Grape, using a number of different architectural views to depict different aspects of the system. It intends to capture and convey the significant architectural decisions, which have been made on the system.

1.2. Scope

This document should be an overview of the whole architecture and the way it should be modeled. Decisions made in this document affect how the system is modeled.

2. 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:

a) Use Case View
b) Logical View
c) Process View
d) Implement View
e) 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.

3. 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:

a) Provide an on-line interactive platform for different users to communicate and share

their resources and opinions.

b) Allow group leaders to track and analyze the effect of their activity.

c) 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:

a) 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.

b) 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.

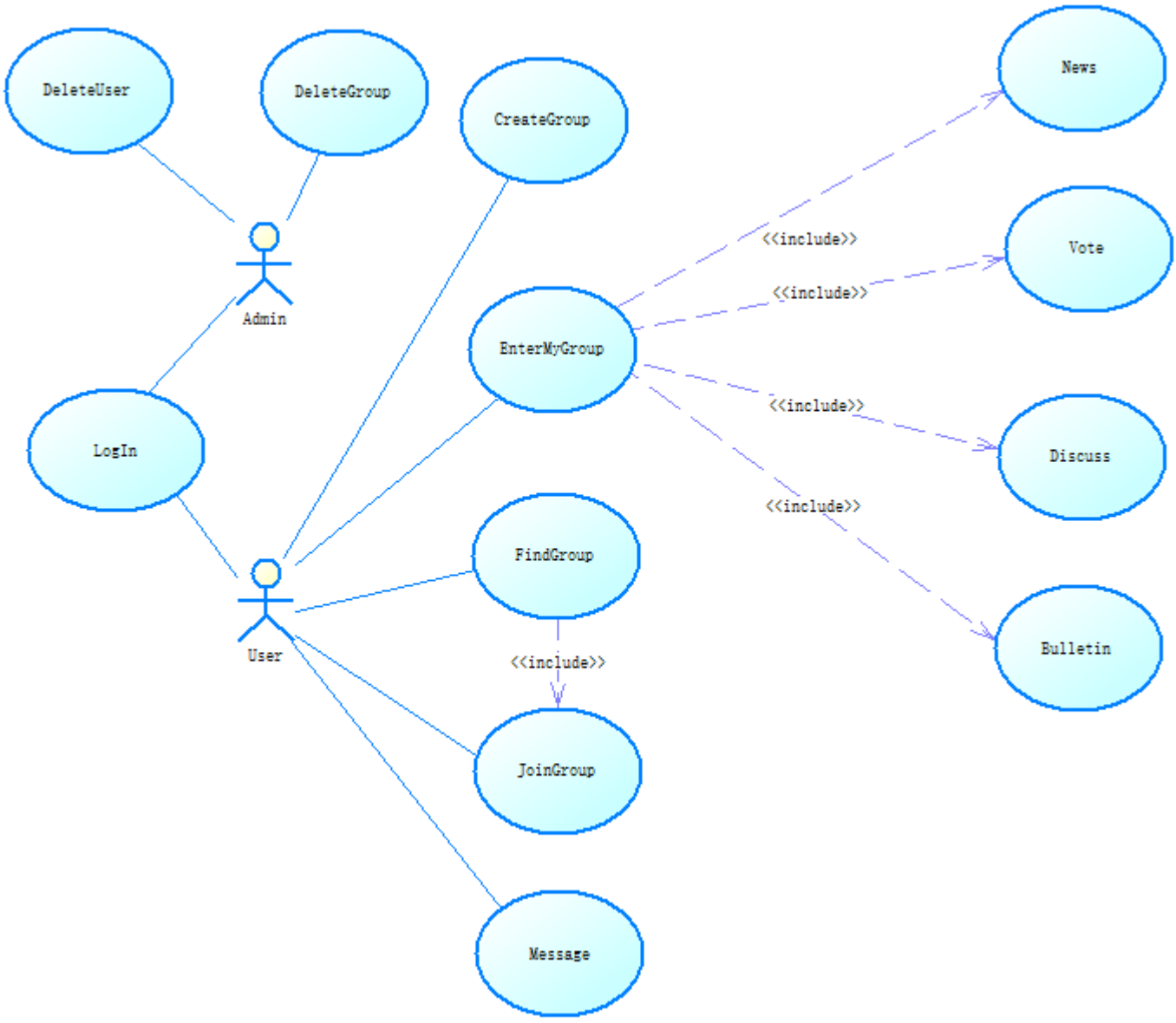
c) All performance and loading requirements, as stipulated in the Vision Document and the Supplementary Specification, must be taken into consideration as the architecture is being developed.

d) 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.

4. Use-Case View

4.1 Overview

This chapter is a main description of the use-case view of the software architecture. The Use Case View is important input to the selection of the set of scenarios and/or use cases that are the focus of iteration. The functionality of Grape is captured in the use-case diagram below:



This architecturally-significant subset of the use cases are described in the following section. These architecture-significant use cases illustrate the key functions of Grape and exercise all major system components. The remaining use-cases can be rapidly developed without changes to the architecture by following the application structure and by reusing the mechanisms described in section 5.

All implemented use cases have an associated Use Case Specification document. References to these documents can be found in the same directory.

4.2 Architecturally Significant use cases

The architecturally-significant use cases are those that "exercise" the most critical parts of the system architecture and demonstrate the core system functionality. In this system they are:

Discuss Operation:

One member can post problems to be discussed by other members in the group.
Basic Scenarios:

1. The system shows the interface for editing the question.
2. The member modifies his question in the form area. Details include a title and the text.
3. The member has finished editing his question, and presses the "submit" button.
4. The system receives the question sent by the member, and returns to the group discussing page.
5. The system reminds all members in the group, and marks the member who asked the question as the owner of this question.
6. To modify the question, the owner can click the "modify" button in the question-viewing page.
7. To delete the question, the owner can click the "delete" button in the question-viewing page.
8. If someone has answered the question, the owner will receive a notion, and can shift to the question-viewing page by clicking the "see details" button.
9. If the owner thinks that he has got a satisfying answer, and the question is no longer needed, he can click on the "delete" button in the question-viewing page. This use case comes to the end.

Vote Operation:

The leader can raise a vote to let the members in the group vote for the choices they select.

a) The group leader selects "New vote".
b) The leader then will be asked whether to raise an "instant vote" or a longlasting vote with more information.
c) If "instant vote" is selected ,then the leader only needs to set a period of time,say,two minutesand attach only one question.If the leader wants to attach more information to the vote,he then needs to add some details such as such as class problems or questionnaires .
d) When the group leader clicks "let's vote", a countdown clock will be displayed.And all the members in the group will be informed of a new vote available, then they have to cast a vote in the limited time the leader set.

When the vote is closed, The result will be shown in the form of histograms or other diagrams.

And the total number of every question the members answered can be seen .

Create group:

One user can be able to create a group to be the leader in the group and there can be new uses to join the group.

Basic Scenarios:

1. The Leader make an application to create a group.
2. The Leader fills out the fundamental blanket to commit some necessary information, including topic, size of group, group name.
3. The Leader invites other Users to the group as Members.
4. Members accept the invitation and join in the group.

Join Group:

The user can have access to be in a group they want.

Basic Scenarios:

The Member searches the group by group name.

1. If the group name exists, system will give all the results. If the group name doesn't exist, system will give some similar results.
2. The Member searches the group by group number.
3. If the group number exists, system will give one certain result. If group number doesn't exist, system will tell the Member there is no corresponding result.
4. The Member can add topic or description attributes to restrict the search results.
5. If the Member find the group, he can send request to the Leader.
6. The Leader receives the Member's request and accepts his attendance.

Bulletin Operation:

The user can receive the information broadcast in the group by leaders.

Basic Scenarios:

1. The system shows the recent bulletin in the given group.
2. The user clicks the bulletin button.
3. The system changes to the bulletin interface and show all the bulletins in this group.
4. The user clicks the "detail" button.
5. The system changes to one of the bulletin and displays the full contents.
6. If the user is the group leader, he can activate the "NewBulletin" operation by clicking the "NewBulletin" button.
7. The new-added Bulletin is included here. At the end of this use case, the system returns to the bulletin interface with a modified bulletin set.
8. If the leader wants to delete an out-of-date bulletin. He can select "delete" button which is

only showed to the group leader besides "detail" button.

Actually, you can reference to the sequence diagrams and collaboration diagrams of these very important use cases in the analysis model.

5. 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:

a) Logical decomposition of the system into three layers.
b) 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:
    a) Front Controller
    b) Command Delegator
    c) Service Locator
    d) Security Handler

2. Data Access and Operation
    a) Persistency
    b) Session Façade

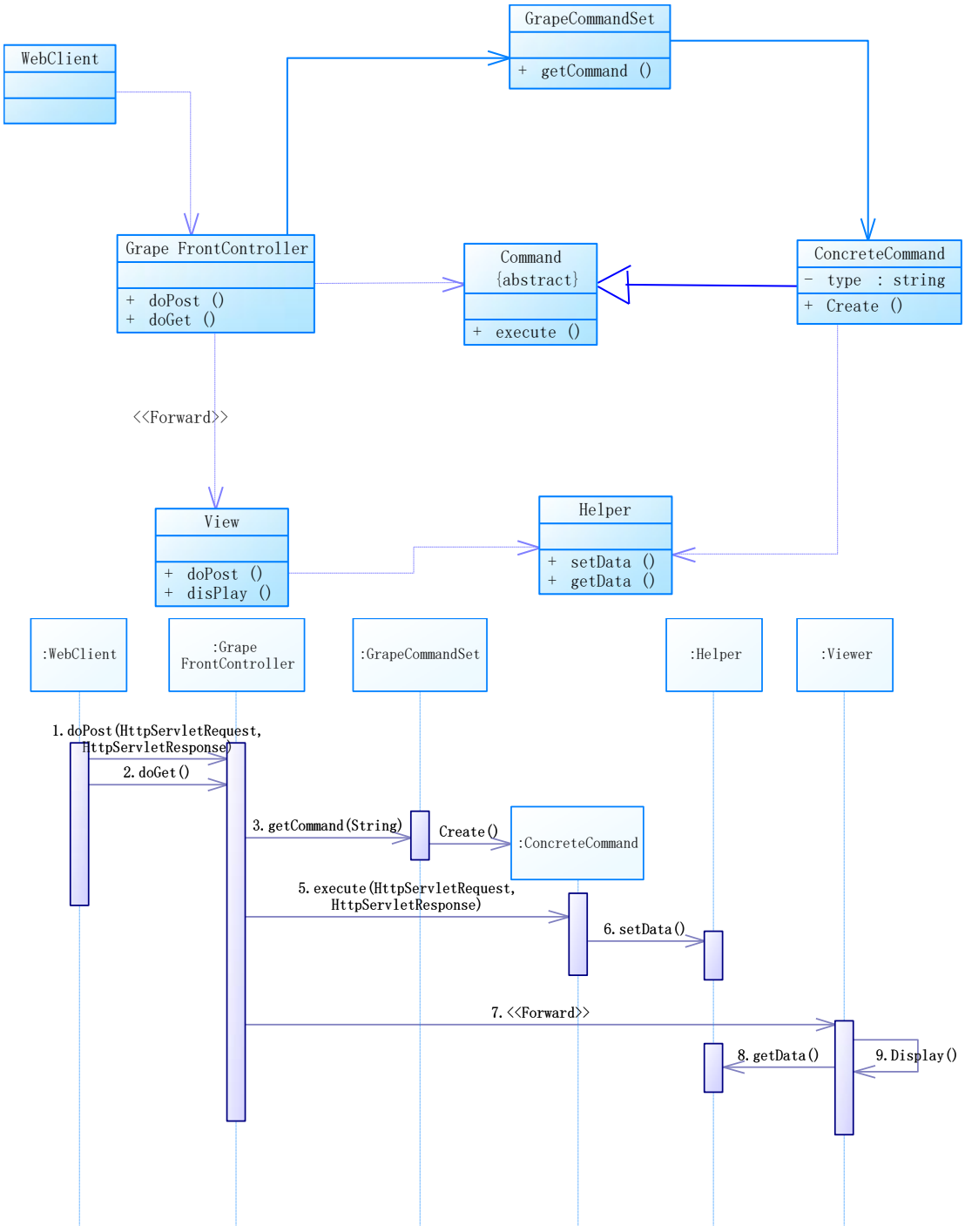
These two kinds of mechanisms will be addressed below. In the following two sections, we will organize each mechanism in a strict and clear order. First, a class diagram and a sequence diagram will be displayed. Then, we will introduce how the mechanism works and the situation we apply it to our Grape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



How it works:

a) The WebClient send his request (maybe: to get data from database), All the requests from multiple Web Browser were send to the Grape FrontController.
b) The front controller inspects each request for consistency and verifies if the user needs to be authenticated and authorized.
c) The front controller analyzes the request and chooses a command that can finish the task. And then get the corresponding command from the command set.
d) The command finished the task if required put some data in the helper. Decide which the next page is to display. The command returns the next page to the Grape FrontController.
e) The Grape FrontController return the next page to the client (in the form of using python flask frame).
f) The view(actually, it is a jsp page) display itself, if needed get some data from the helper.

Key point:

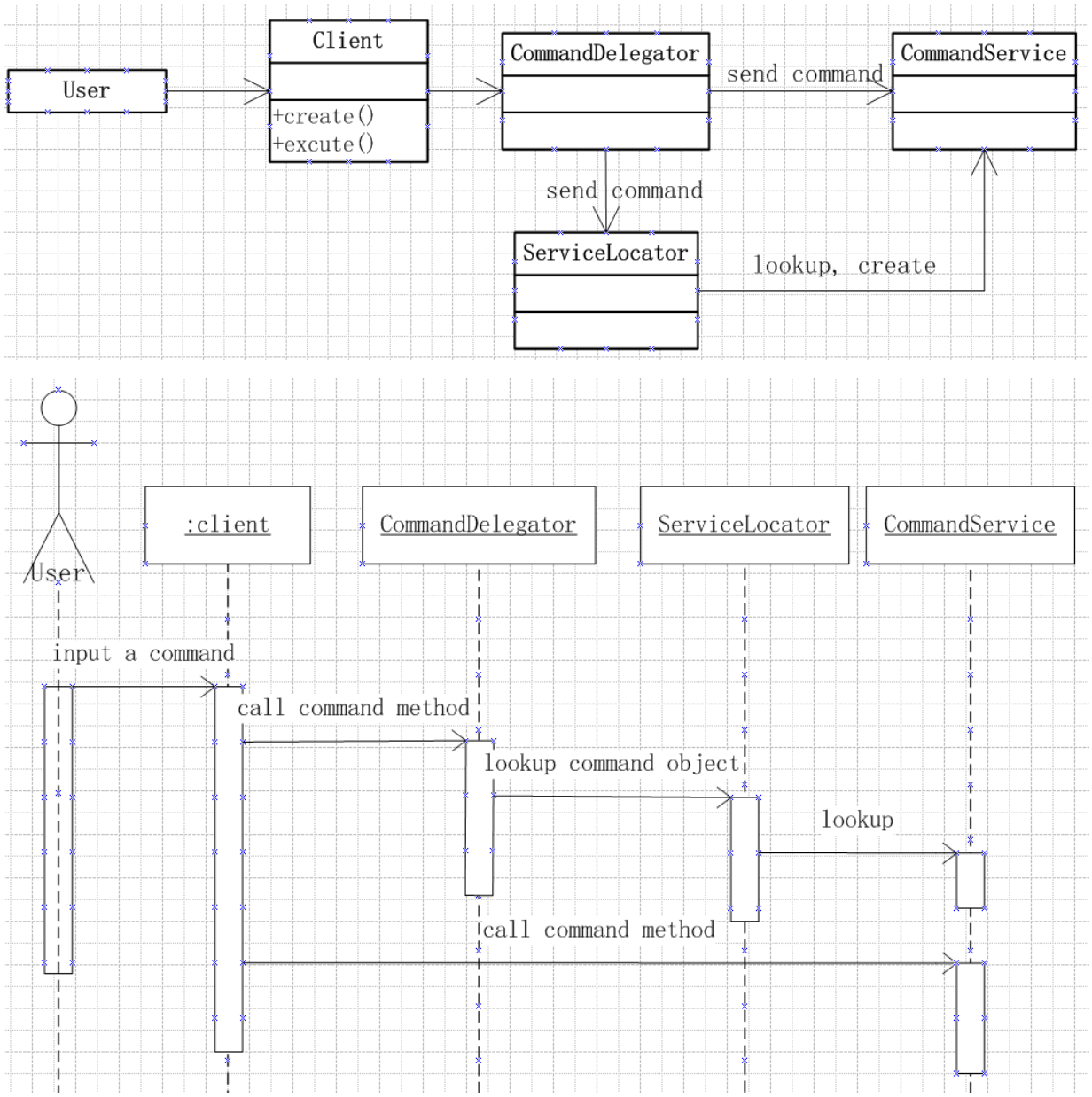
a) The front controller is implemented as a HttpServlet.
b) We generate different pages according to different commands by change the code of html template and the parameters in it.
c) Command and Object Factory design patterns are used here.

Advantages:

a) The Page designer doesn't need to concern about the logic. He only care about how to present data in valuebeans gracefully.
b) A page never needs to know which page is next to present if the form is submitted or a URL is clicked. All the requests from users were sent to front controller. So pages never need to know how other page is designed.
c) The command pattern decreases coupling. The front controller simply uses command to finish the logic, never need to know other classes.
d) Page template and command can be developed separately. We use flask frame to insert the corresponding parameters into the corresponding page templates.
e) A controller manages business logic processing and request handling. Centralized access to an application means that requests are easily tracked and logged.
f) A controller promotes cleaner application partitioning and encourages reuse, as code that is common among components moves into a controller or is managed by a controller.

5.2.2. Command Delegator

Class diagram and sequence diagram:



How it works:

A client (in most cases it is a command) that requires access to a business service component creates an instance of a session delegate called Command Delegator.

Key Point:

Usually the commanddelegate is used with a session facade; typically there is an one-to-one relationship between the two.

A CommandDelegator uses a component called the 'lookup service' (ServiceLocator). The 'lookup service' (ServiceLocator) is responsible for hiding the underlying implementation details of the business service lookup code. (The ServiceLocator mechanism will be depicted later.)

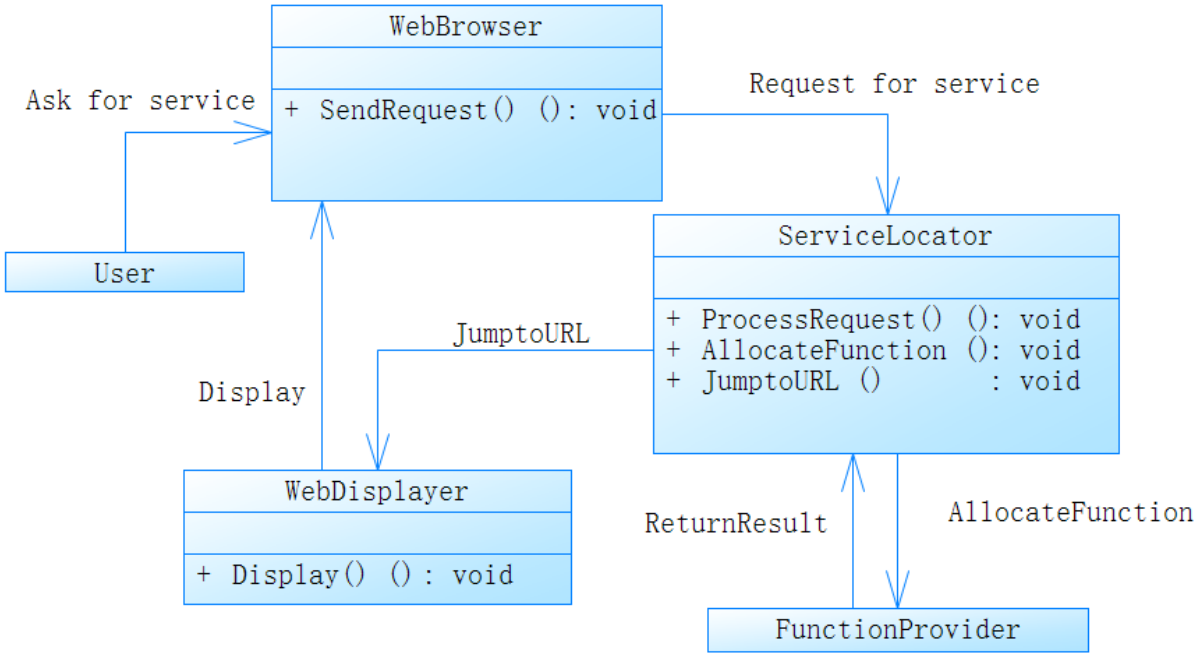
Advantages:

a) Use a CommandDelegator to reduce coupling between presentation-tier clients and business services. The CommandDelegator hides the underlying implementation details of the command service, such as lookup and access details of thearchitecture.

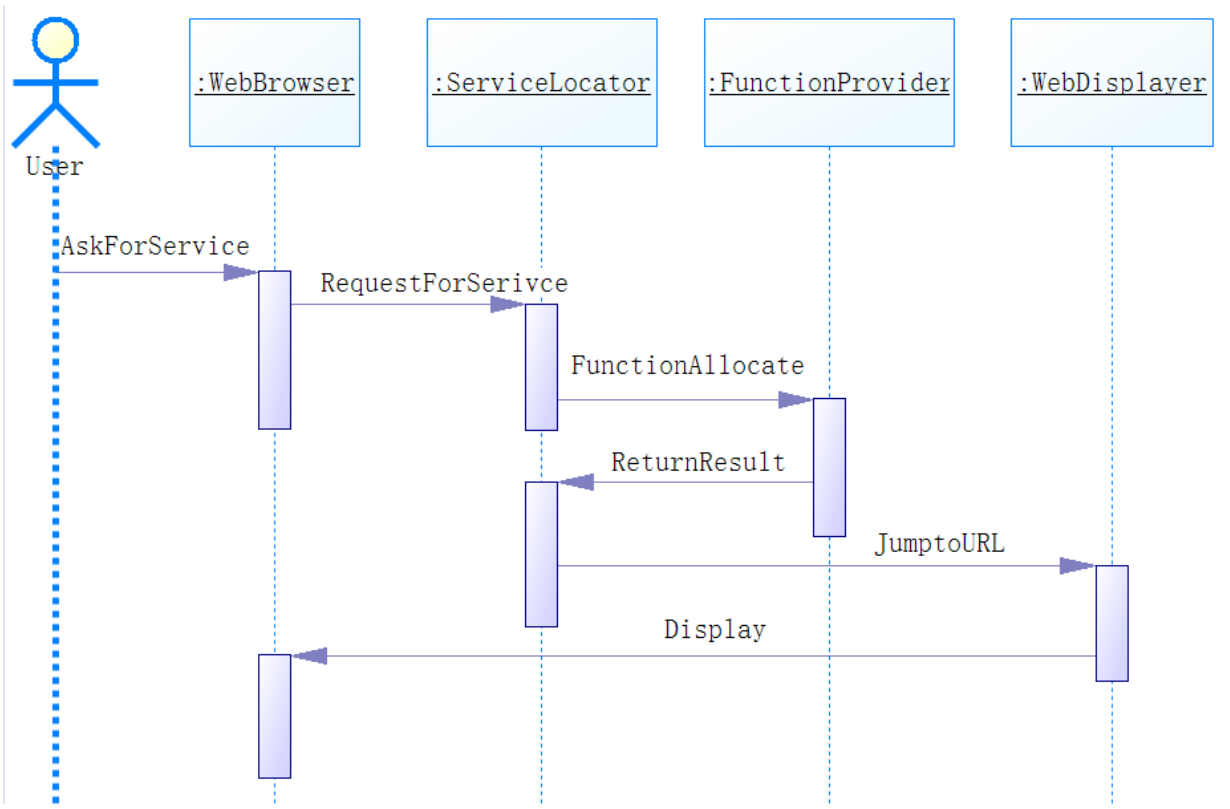
b) If necessary, the delegate may cache results and references to remote command services. Caching can significantly improve performance.

5.2.3. Service Locator

Class Diagram:



Sequence Diagram:



Note:

The Function Provider is a collection of functions which will be used further. We organize all functional operations in this file in order to make codes orthogonal and less coupled.

The Web Displayer is the corresponding html web page in order to display new information in web browser.

How it works:

Since the user can only see the web page in web browser, the user can only operate on the web browser interface. When the user interacts with the web page, the web browser invoke the Service Locator to request for new information. The Service Locator analyze the operation and accordingly invoke the corresponding functions and then invoke the web displayer to display new information.

Key Point:

The Service Locator receives requests from web browser (actually from the operation from user), and processes the request in the corresponding area, the process involves requesting new data from Function Provider. After getting new data, the Service Locator locates the corresponding web displayer to display the new data.

Advantages:

a) The Service Locator pattern encapsulates the complexity of this interactive process

andcreation process (described in the problem) and keeps it hidden from theclient.

b) It provides a very useful and precise interface that allclients can use. The pattern interface ensures that all types of users in theapplication uniformly access back-end objects, in terms of requests. This uniformity reduces development and maintenance overhead.

c) Because users of Grape system are not aware of t Service Locator he Service Locator objects, it's possible to add new Service Locator objects for our system developed anddeployed at a later time without impacting the users.

d) The users are not involved in Function Provider process, which is hidden two-level deep from users.. Because the Service Locator performs this work, it can aggregate thenetwork calls required to operate on different objects.

5.2.4. Security Handler

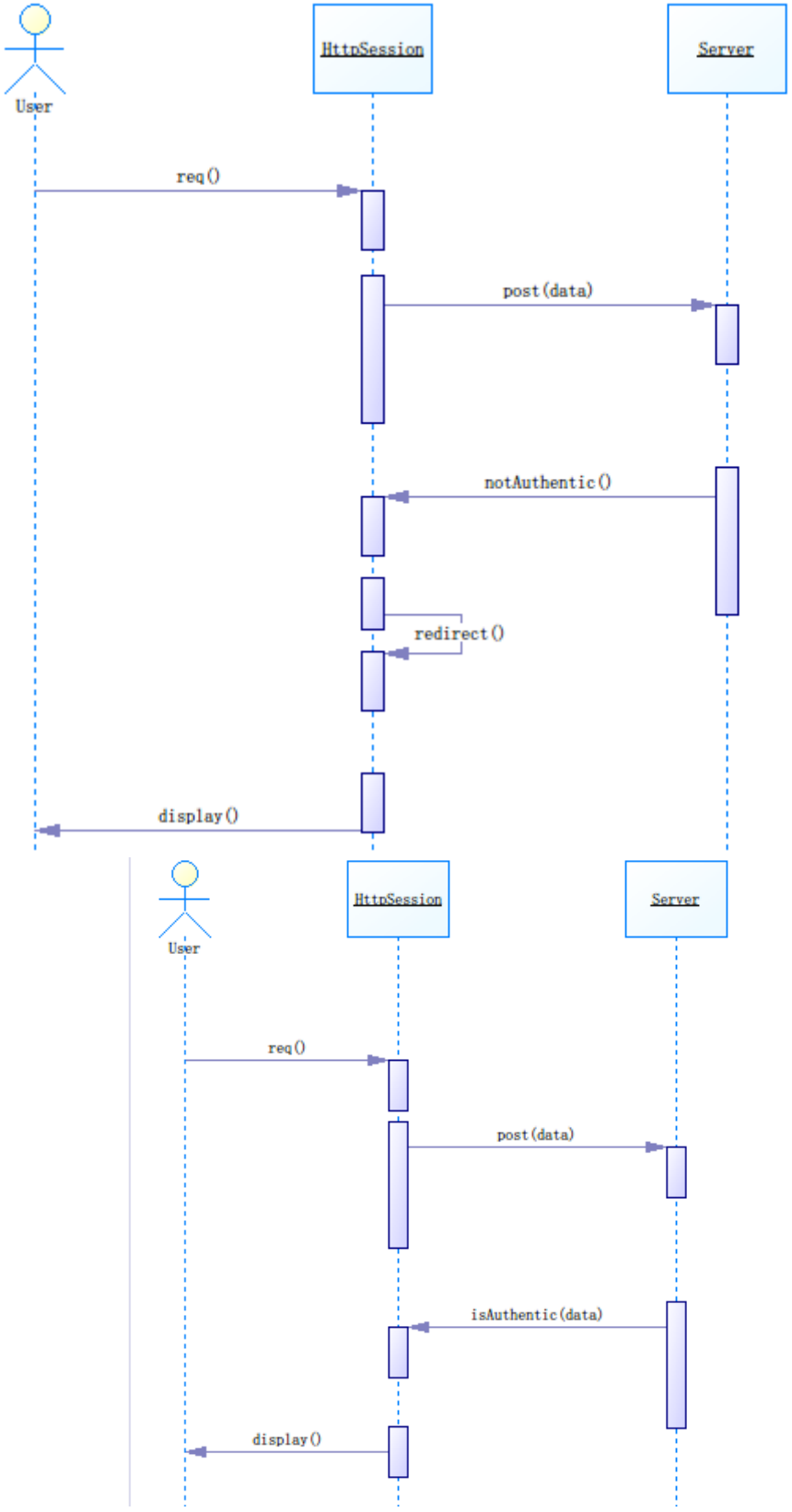
How it works:

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:

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

Sequence diagram:



5.3. Data Operation Mechanisms

5.3.1. Persistency

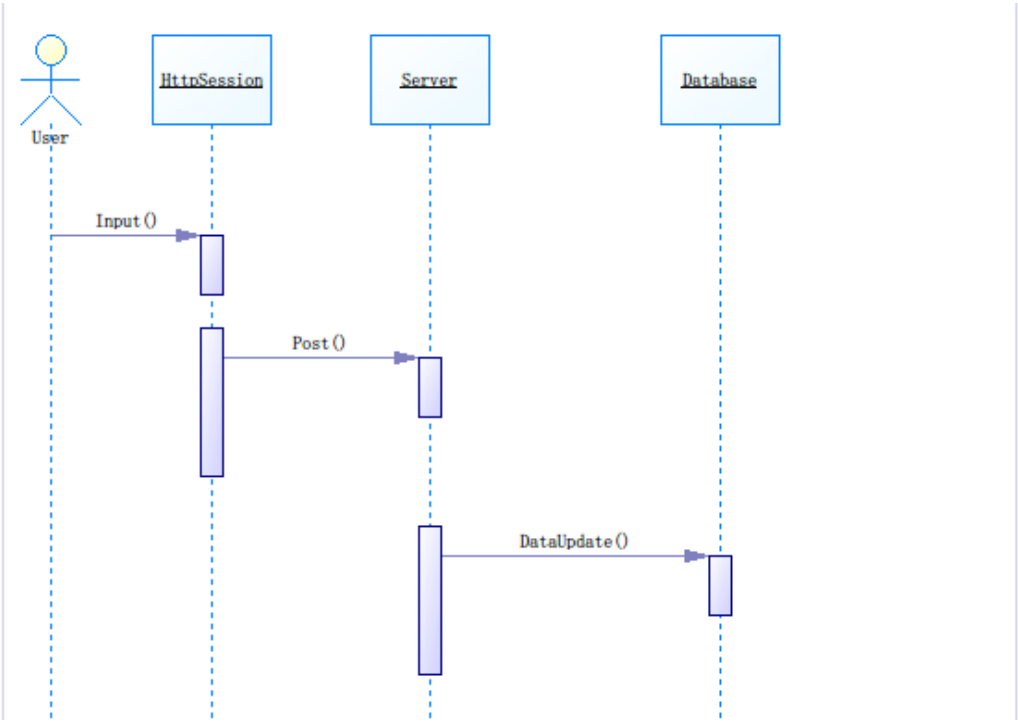
How it works:

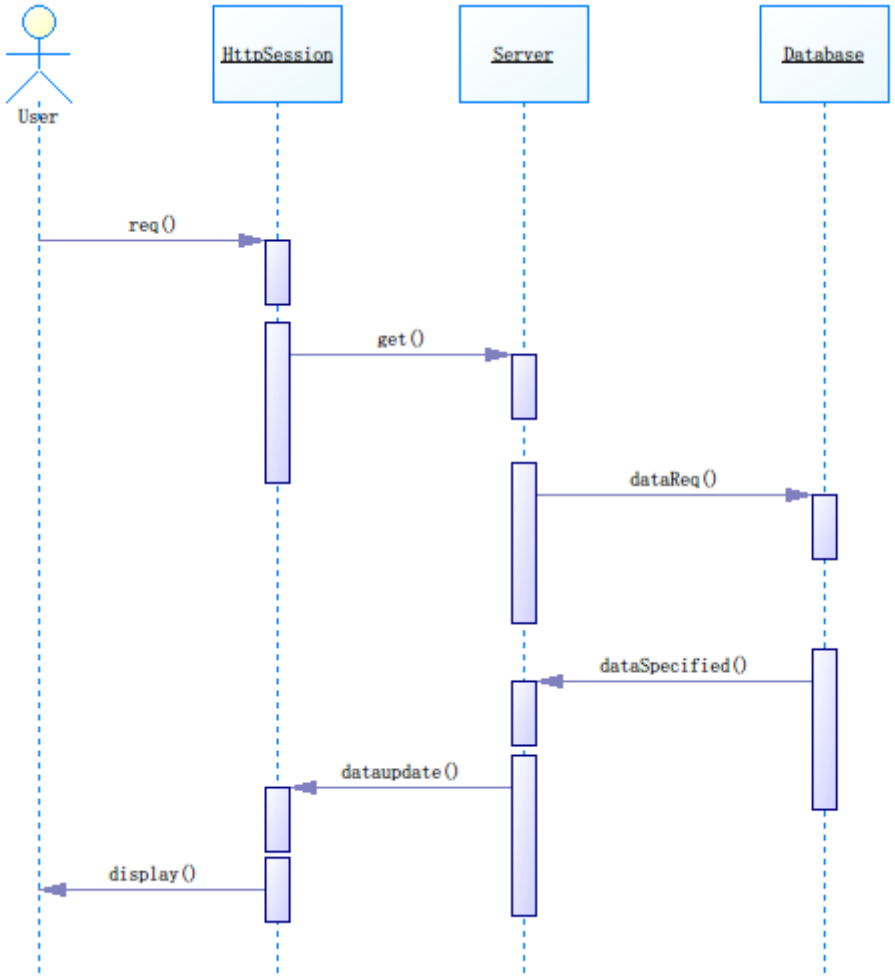
In our system, we use mysqldb for python to connect MySQL database. One of the convenience is that you we can use the same SQL instruction in python by mysqldb. And it's therefore simple to operate dynamic change of all kinds of data.

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 everyone 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:

a) All business service components are implemented as Session EJBs or have Session EJB façades.
b) There is always no one-to-one relationship between Session Facade and the Entity beans. In this system how Session Facade Manage its Entity Beans will be depict below as Architecturally-significant Model elements.
c) In this system, a Business Facade is used by a Business Delegator directly.

Advantages:

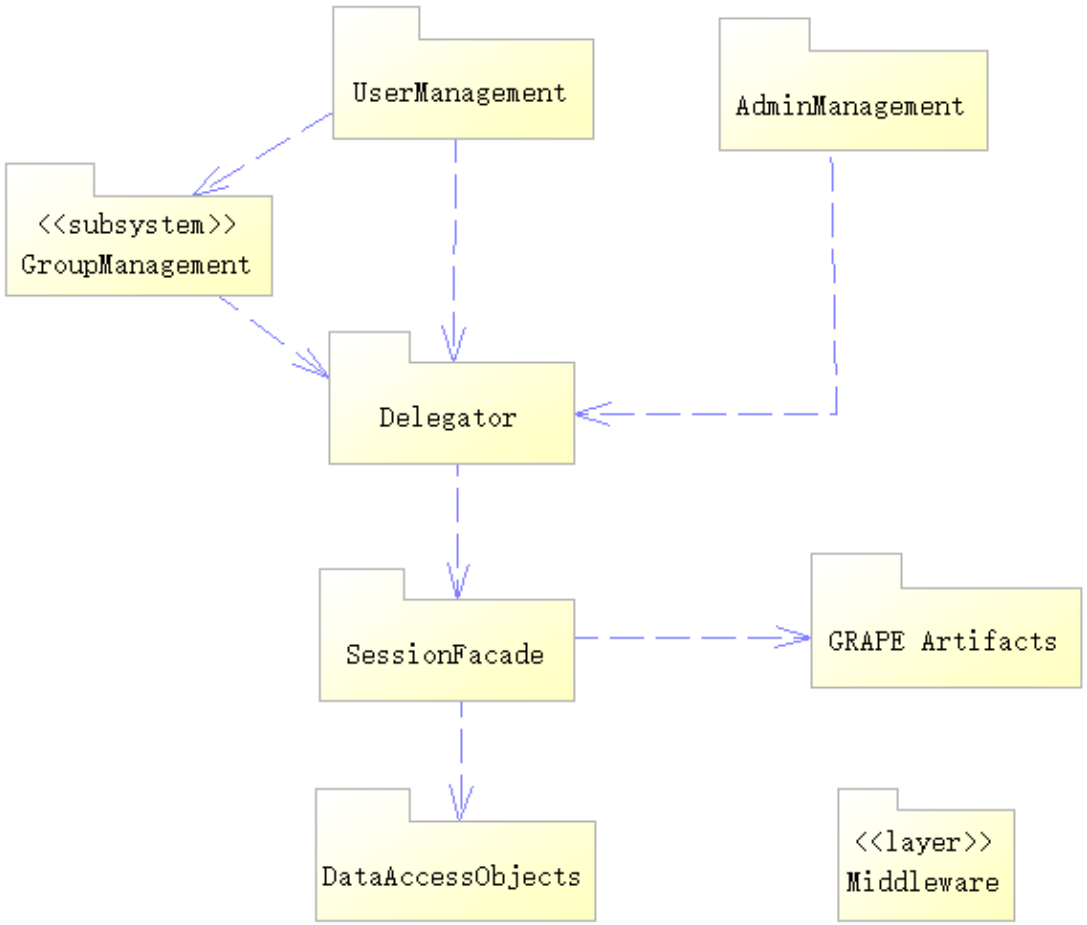
a) Session Facades can represent a control layer between clients and the business tier, as identified through analysis modeling.
b) The underlying interactions between the business components can be very complex. A Session Facade pattern abstracts this complexity and presents the client a simpler interface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

5.4. Architecturally Significant Use Case Realization

Addressed in the design model, please refer to it.

5.5. Architecturally Significant Model Elements

Architecturally-Significant Packages:



Admin Management: Admin Management related classes, admin operations must be put in this package.
User Management: User Management related classes, user operations must be put in this package.
GroupManagement Subsystem: Group Management related classes, group operations must be put in this package.
Delegator: business delegate classes
Session Facade: business session façade.
Grape Artifacts: python, flask
Data Acces Object: Data Access Objects, see mechanism in 5.2.

5.6. Architecturally Significant Classes

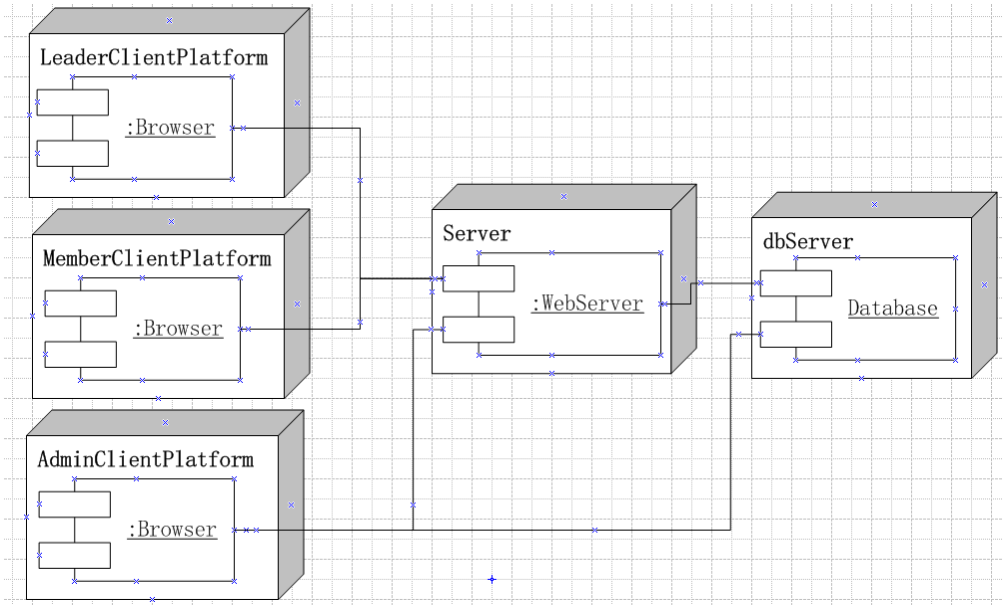
Front controller: see front controller mechanism in 5.2.
Abstract Command: see front controller mechanism in 5.2.
Abstract Command Factory: see front controller mechanism
Command Factory Impl: see front controller mechanism in 5.2
Security Handler: see Security mechanism in 5.2.
Error page: see Security mechanism in 5.2.

All of the above exist in the Application layer.

Service Locator: See service Locator Mechanism in 5.2.
Subsystem interfaces: I) authorities manager and I) staff infosystem.

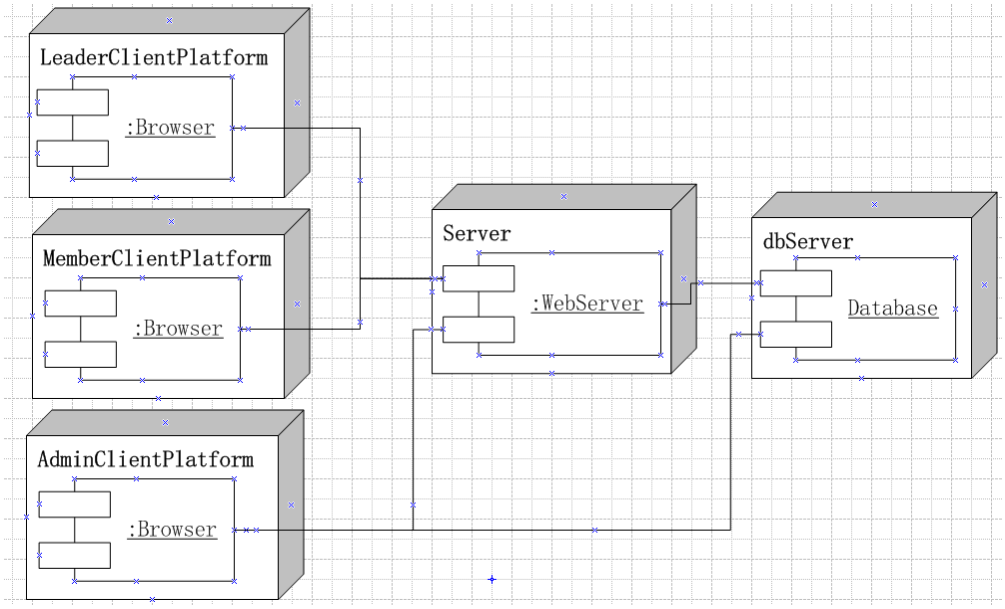
6. Deployment View

The deployment view of a system shows the physical links between different nodes when the system works. Grape basically runs on a web server, with an dbServer providing access to data of users. Users can access to grape with a browser, while an admin can have access directly to the database.



7. Implementation View

The deployment view of a system shows the physical links between different nodes when the system works. Grape basically runs on a web server, with an dbServer providing access to data of users. Users can access to grape with a browser, while an admin can have access directly to the database.



8. Size and Performance

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

a) 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.
b) The system shall provide access to the legacy course catalog database with no more than 10-second latency.
c) The system must be able to complete 80% of uploading operations in at most 2 minutes.
d) The system must be able to complete 80% of the downloading operations in at most 1 minutes.

9. System Size

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

a) Labor months: 5
b) Business components: 5
c) Dependencies on external components: 4
d) Lines of total coding: 8,000
e) Source file number: 100
f) 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.