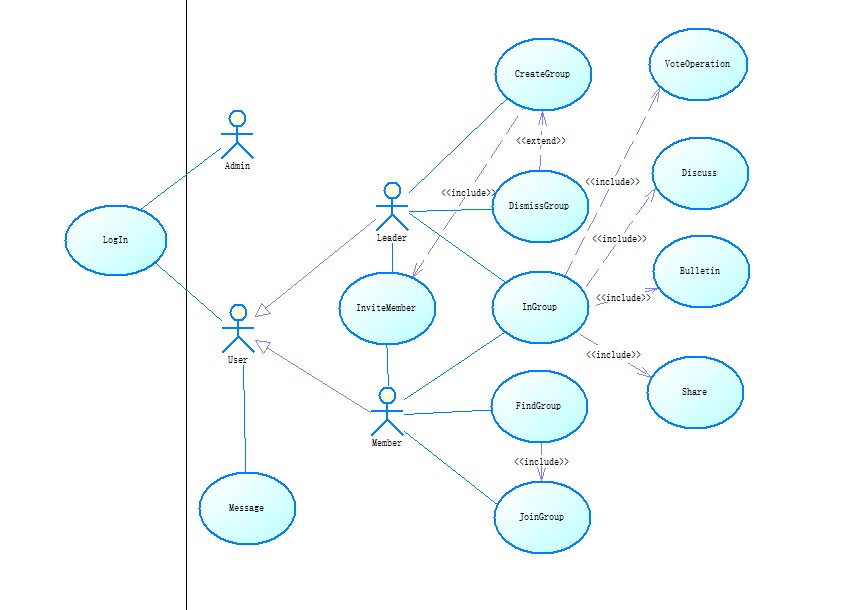
4. Use Case View

A 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 an iteration. The functionality of Grape is captured in the use-case diagram below:



These architecturally-significant subset of the use cases are described in the following section. These architecture-significant use cases illustrate the key functions of BTS 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.

**4.1 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:

**Share**: This use case allows members to upload files to share files with other members in the group.

Basic Scenarios:

1. The member selects “Share Files” to open the sharing files panel.
2. The member press “Files” to enter the file system on his terminal.
3. The member can select the files he want to share. Then the application will return to the sharing files panel with the filenames and sizes displayed on the screen.
4. The member press “submit” button to upload the files, then the file upload begins and a process bar is showed on the screen.
5. When the process bar is full which means the upload is finished. The member press “Finished” and the process bar dismisses, and then back to the sharing panel. Then the system reminds the leader at first.
6. The leader review the files and choose whether the files should be shared or not. If yes, he can press “permit” to make them shared in the group. If no, he can press “deny” to delete the files and system responds to it by deleting the files on the server.
7. If the leader has pressed the “permit” button, system would remind all members included in the group and show the filenames and sizes on the download panel.

**Discuss**: 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.

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

1. The group leader selects “New vote”.

2. The leader then will be asked whether to raise a “quick-vote” or a vote with some information.

3. If “quick-vote” is selected ,then the leader only needs to set a period of time,say,two minutes.If the leader wants to attach some information to the vote,he then needs to add some details such as class problems or the self-information of the candidates.There are entries like title,description and attachment.

4. When the group leader clicks “Start”, 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.

5. When the vote is closed, The result will be shown in the form of histograms and other diagrams. And every choice the member made can be seen in detail by the group leader.

6. If the leader allowed the members to see the result then all members in the group can choose whether to save the result or not.. The default option is save.Otherwise only the

**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**: 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 “ModifyBulletin” operation by clicking the “modify” button.
7. The ModifyBulletin is included here. At the end of this use case, the system returns to the bulletin interface with a modified bulletin set.

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

**5.2.1Front Controller:**

Class diagram and sequence diagram:





**How it works:**

1. The WebClient send his request (maybe: to get data from database), All the requests from multiple Web Browser were send to the Grape FrontController.
2. The front controller inspects each request for consistency and verifies if the user needs to be authenticated and authorized.
3. 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.
4. 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.
5. The Grape FrontController return the next page to the client (in the form of using python flask frame).
6. The view (actually, it is a jsp page) display itself, if needed get some data from the helper.

**Key point:**

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

**Advantages:**

1. The Page designer doesn’t need to concern about the logic. He only care about how to present data in value beans gracefully.
2. 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 a page never need to know how other page is designed.
3. The command pattern decreases coupling. The front controller simply uses command to finish the logic, never need to know other classes.
4. Page template and command can be developed separately. We use flask frame to insert the corresponding parameters into the corresponding page templates.
5. A controller manages business logic processing and request handling. Centralized access to an application means that requests are easily tracked and logged.
6. 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.