Grade Central

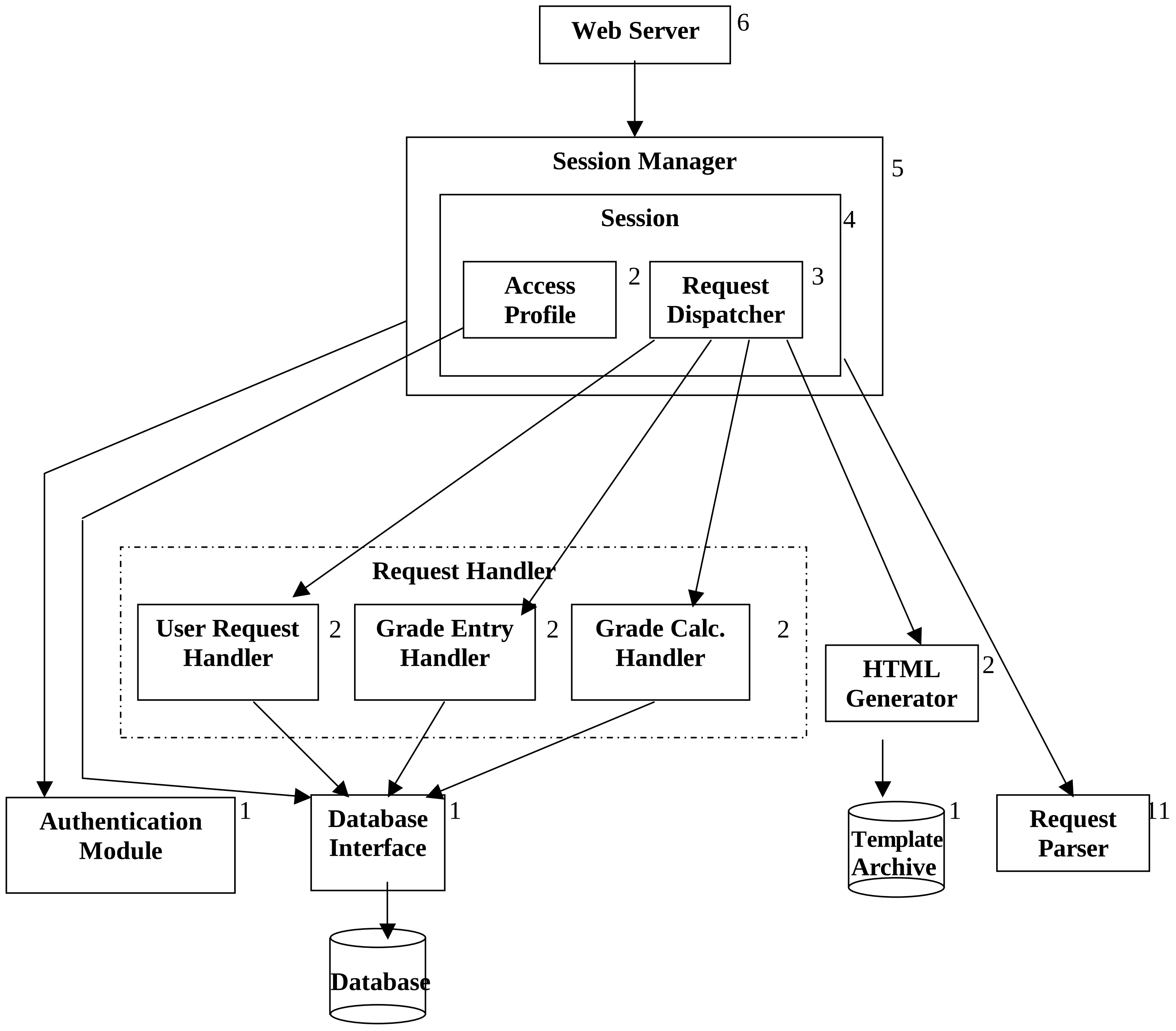
A Campus Grading Tool

Final Design Document

March 9, 2001

1 System Diagram

Each component is labeled with its level rank.



Component Descriptions

Session Manager

This server has a list of ***Session*** objects. When a request comes in, via port 80, from a particular user for the first time, it verifies the user using a third party authenticator which may be Kerberos, or a password file validation system. If the user is properly authenticated, the SM then creates a ***Session*** object for that user and put it in its *SessionList*. When subsequent messages come in, it looks for the session for that user. The *SessionList* will go through garbage collection at regular intervals to eliminate stale sessions.

In the first phase implementation, this component will be single threaded. If success is found and time allows, this will be changed to a multi-threaded approach.

Session Class

The first job of this object is to use the **Request Parser** to decipher the CGI request from the user. The response is in the form of an OpCode. Once it knows the desired action, it much check to see if the user has the privilege to take such an action.

This session object has an **Access Profile** object in it. This object maintains the list of privileges that this user has for specific classes – ie. Prof of Course XX345, or TA for course YY873. This object will check the OpCode for the class in question.

Each session object also contains a **Request Dispatcher**. The dispatcher uses the OpCode previously returned from the **Request Parser** and decides which component method in the RequestHandler to call. When the Request Handler method returns, the Dispatcher calls the HTML generator and sends it the Datablock in order to make the next page output. The page returned from the HTLM generator is returned to the **Session Manager**.

Public Methods

Constructor(string userName)
ProcessRequest(string Request) – returns an HTML string

Request Parser

This string parsing module breaks down the CGI request in order to pinpoint the desired operation. It returns an OpCode which is defined in a table to specify a particular operation.

Public Methods

parseRequest(string Request, int &OpCode, string &course, vector & paramPairs)
– returns a bool and &

Access Profile

This class obtains user information from the database. It stores the courses and privileges of the particular user.

Public Methods

Constructor(string userName) – Obtains initial list of privileges
hasAuthority(string course, int OpCode) – returns a boolean value

User Request Handler

This module handles all administrative changes to the database. This includes user and course changes and additions. Each of these methods make calls to the database.

Public Methods

AddUser(string course, vector paramPairs) returns boolean
AddCourse(string course, vector paramPairs) returns boolean
AddStudent(string course, vector paramPairs) returns boolean
AddProfessor(string course, vector paramPairs) returns Boolean
MakeTA(string course, vector paramPairs) returns boolean
EditUser(string course, vector paramPairs) returns Datablock
EditCourse(string course, vector paramPairs) returns Datablock
EditStudent(string course, vector paramPairs) returns Datablock
EditProfessor(string course, vector paramPairs) returns Datablock
RemoveUser(string course, vector paramPairs) returns boolean
RemoveCourse(string course, vector paramPairs) returns boolean
RemoveStudent(string course, vector paramPairs) returns boolean
RemoveProfessor(string course, vector paramPairs) returns Boolean
RemoveUser(string course, vector paramPairs) returns Boolean
RemoveTA(string course, vector paramPairs) returns Boolean
ViewClassList(string course, vector paramPairs) returns Boolean

Grade Entry Handler

This module handles all grade information. This includes assignment and grade changes and additions. Each of these methods make calls to the database.

Public Methods

AddAssignment(string course, vector paramPairs) returns Datablock
AddCourseGrades(string course, vector paramPairs) returns Datablock
WeightAssignments(string course, vector &paramPairs) returns Datablock
EditAssignments(string course, vector & paramPairs) return Datablock
ViewCourseGrades(string course, vector &paramPairs) return Datablock
EditCourseGrades(string course, vector &paramPairs) return Datablock

Grade Calculation Handler

This module handles all grade calculations. This includes averaging a particular assignment, averaging a student's grades, and creating a histogram for an aassignment.

Public Methods

AverageAssignment(string course, vector paramPairs) returns Datablock
AverageStudent(string course, vector paramPairs) returns Datablock
CreateHistogram(string course, vector &paramPairs) returns Datablock

HTML Generator

This class uses the parts of pages in the template library to construct a web page given the information from Request Handler.

Public Methods

```
void parseTemplates();

//returns the page with the specified filename:
HTMLPage pageWithName(String filename);

//alternately, returns the initialized page with the specified filename:
HTMLPage pageWithNameAndParem(String filename,
```

Template Library

This library contains parts of pages that have been designed. IT may contain a few headers, or footers, or pieces of body, which the HTML generator then strings together – filing in any information.

Database Interface –

This handles all reading from and writing to the database. It takes for input, and returns for output, Datablocks. This component handles all input to and output from the database. It serves as an abstraction from the actual database. Other functions that use the need not use SQL to access data.

Public Methods

Update Database:

makeNewCourse(course code, professor name)
removeCourse(course code)
setProfessor(course code, professor name)
setSchema(course code, schema)
addStudent(student name, course code, grade option)
removeStudent(student name, course)
setGradeOption(course code, student name, grade option)
addTA(ta name, course code)
removeTA(ta name, course code)
addStudentToTAGroup(student name, ta name)
removeStudentFromTAGroup(student name, ta name)
addAsgnToTA(course code, asgn name, ta name)
removeAsgnFromTA(course code, asgn name, ta name)
addTypeToTA(course code, type name, ta name)
removeTypeFromTA(course code, type name, ta name)
addAllToTA(course code, ta name)
removeAllFromTA(course code, ta name)
addAssignment(course code, asgn name, type name, weight, total possible)
removeAssignment(course code, asgn name)
setAssignmentType(course code, asgn name, type name)
setAssignmentWeight(course code, asgn name, weight)
setAssignmentTotal(course code, asgn name, total possible)
addType(course code, type name, weight, distribution)

removeType(course code, type name)
Update Database (cont):
setTypeWeight(course code, type name, weight)
setTypeDistribution(course code, type name, distribution)
addGrade(course code, asgn name, student name, grade, lateness, comment)
setGrade(course code, asgn name, student name, grade)
setLateness(course code, asgn name, student name, lateness)
setComment(course code, asgn name, student name, comment)
setFinalGrade(course code, student name, grade)

Request Data:
getProfessor(course code) {returns professor name}
getSchema(course code) {returns grading schema}
getGradeOption(student name, course code) {returns grade option}
getTAofStudent(course code, student name) {returns ta name}
getAssignmentType(course code, asgn name) {returns type name}
getAssignmentWeight(course code, asgn name) {returns assignment weight}
getAssignmentTotal(course code, asgn name) {returns total possible}
getTypeWeight(course code, type name) {returns type weight}
getTypeDistribution(course code, type name) {returns distribution}
getGradeValue(course code, asgn name, student name) {returns grade}
getGradeLateness(course code, asgn name, student name) {returns lateness}
getGradeComment(course code, asgn name, student name) {returns comment}
getFinalGrade(course code, student name) {returns final grade}
getAllStudents(course code) {returns list(student name)}
getAllTAs(course code) {returns list(ta name)}
getAllAssignments(course code) {returns list(assignment name)}
getAllTypes(course code) {returns list(type name)}
getStudentsInGroup(course code, ta name) {returns list(student name)}
getAssignmentsAuthorizedForTA(course code, ta name) {returns list(assignment name)}
getAuthorizedTAsForAssignment(course code, asgn name) {returns list(ta name)}
getDefaultTAsForType(course code, type name) {returns list(ta name)}
getAllAssignmentsOfType(course code, type name) {returns list(assignment name)}
getAllRoles(user name) {returns list(course, authority)}
getTAsOfAllStudents(course code) {returns list(student name, ta name)}
getAllGradesOnAssignment(course code, asgn name)
{returns list(student name, grade, lateness, comment)}
getAllGradesOf Student(course code, student name)
{returns list(assignment name, grade, lateness, comment)}
getAllFinalGradesInCourse(course code) {returns list(student name, grade)}
getFinalGradesInAllCourses(student name) {returns list(course code, grade)}

Web Server

This is a server, listening on port 80, for requests to come in. When they do, it passes along the requests to the **Session Manager.**

Authorization Module

This module, called by the Session Manager verifies the user by securely checking the password. Kerberos may be used for this functionality.

2 Functional Specifications (with priorities)

This section lists all requirements of the system. The number after each item is its implementation priority (see schedule; 1 = Feature Set 1, 2 = Feature Set 2, 3 = “Perhaps if we have extra time”).

2.1 User Interface Requirements

- Grade Central must have an intuitive GUI interface. (1)
    - The flow of control of the user’s commands must be designed and tested such that at each step, the user is presented with every action that logically follows from the action just taken, and no illogical actions are presented.
    - Grade entry screen shows small group of students at a time instead of 1 large scrolling table of 100 students. (1)
    - Allows for different grading schemes – numerical, letter grade, etc. (1)
      (Note: “calculation” of letter grades assumes a 4.0 scale; professors may disregard the calculation feature entirely and assign final grades based on own assessment.)
    - If one assignment is deleted, a DISTRIBUTE button would evenly redistribute that percentage among others. (3)
- The program must provide helpful documentation, accessible at any point. (1)
    - A library of help documents, with a table of contents and index, will be stored as a set of HTML pages on the server. (1)
    - Access to these help files will be provided through a button or link present on every screen with the same location and appearance. (1)
    - The help library will be searchable by keyword. (3)
- The program must be web based to allow accessibility by different platforms and from any location with a web hookup (1)
- The web interface must provide a secure connection with the back-end server (i.e. use HTTPS). (1)
- The model of interaction between the user and server must conform to users’ familiarity level with similar systems, to facilitate learning to use this system. (1)
  What this actually implies is the requirement that requests from the user be treated independently, and that the back-end server make no assumptions about the state of the user’s display and/or actions outside of the current request.
- Must comply with Netscape 4.0 HTML standards. (2)

2.2 Security Requirements

- User must log into system with a username and password (1)
- When authenticated, the user must only be allowed to access data that he/she has been authorized to access (1)
- There are four types of access roles for viewing/entering data (1)
    - Administrator
    - Professor
    - TA (Teaching Assistant)
- Security and access authority will only be granted by an Administrator (for Professors) or a Professor (for a TA) (1)
- A timeout value will be assessed for grade entering, in case the user leaves the terminal unsecured. (1)
- The actual username-password authentication mechanism will be hidden behind a façade interface, to minimize dependency on any particular external authorization method. (1)

2.3 Database-Specific Requirements

The following information must be stored about each user:

- Username (1)
- Course Affiliation (1)
- User Type – Registrar, Professor, TA, Student (per course) (1)
- Enrollment Status(2) – Enrolled, Dropped, Auditing
- Last Name (1)
- First Name (1)
- Nickname (2)
- Email address (2)

2.4 User Actions: Functionality-Specific Requirements

- Administrator
  - Create courses (2)
  - Create users (2)
  - Edit users' course affiliations and any roles (1)
  - Import class list data from file (3)
  - Export grade data to file (3)
- Professor
  - View all users affiliated with course (1)
  - Add course affiliation and Student/TA roles to existing users (1)
    (implicit in the "add student to course" action)
  - Create users and assign them Student and/or TA roles (1)
    (implicit in the "add student to course" action)
  - Create class subgroups one level deep (3)
    (e.g. lab groups)
  - Create subgroup affiliations for users already affiliated with the class (3)
  - Search for a user affiliated with a certain course
    - By username (1)
    - By first name (1)
    - By last name (1)
    - By role (2)
    - By nickname (2)
    - By email (2)
    - Using wildcards (2)
  - Add assignment to course (1)
  - Set assignment weighting (1)
  - Create assignment types (e.g. lab, homework) and assign group affiliations to individual assignments (2)
  - Set weights for types of assignments (2)
  - Set weights per assignment within types (3)
  - Enter grade per student per assignment, in batches (1)
    - Simultaneously enter lateness and text comment per student per assignment (2)
    - After all numerical grades for a particular assignment are entered, software displays a histogram and allows the professor to assign ranges for A, A-, etc. (3.9)
  - Tell system to calculate grade average for each user (1)
  - Tell system to calculate grade average (across course) for each assignment (2)

  - Edit final letter grade for each student (2)
  - Email to all students that grading is done (3.9)
  - Export an archival copy of grade-book for future reference. (3)

- TA
  - View all users affiliated with course (1)
  - Search for a user affiliated with a certain course
    - By username (1)
    - By first name (1)
    - By last name (1)
    - By role (2)
    - By nickname (2)
    - By email (2)
    - Using wildcards (2)
  - Enter grade per student per assignment, in batches (1)
    - Simultaneously enter lateness and text comment per student per assignment (2)
  - Tell system to calculate grade average for each user (1)
  - Tell system to calculate grade average (across course) for each assignment (2)

- Student
  - View assignment grades for a particular class (1)
  - View final grades of all classes at end of term (1)

3 Performance and Reliability Requirements

- During normal operation, response time for any key click must be within 10 seconds. The use of frames may aid in speeding up web transfers.
- During peak operation, response time must be within 20 seconds.
- Grade information stored on server must be available at all hours.
- Databases must be backed up daily – using a snapshot method as to avoid inaccessibility. (rely on existing system backups for development process)

4 Testing Strategy

The primary purpose of testing is to detect any deficiencies in the project so that the final product will be of the highest quality. In order to accomplish this, the project team must aggressively integrate testing into all aspects of the project. Changhee Pyo and Imeh Williams will coordinate testing efforts and work with fellow group members during component, integration, and user testing.

4.1 Component Testing

A bottom-up testing style will be used for component testing, because it complements the implementation process; components with a lower dependency are implemented before components with a higher dependency. For example, the database interface/database and HTML generator will be implemented and tested first since they are not dependent on any other component. The goal of component testing is to ensure that each individual component is functional according project guidelines. Each testable component will have accompanying test programs written with the help of the individual implementing the component. Whenever the implementer reaches a milestone with his/her component, test code will automatically test the functionality and provide a progress report for the implementer. The primary responsibility of the implementer is to help develop test data and provide information about scenarios that might undermine the component's functionality. The following is a summary of how the various components might be tested:

Database Interface: The database managers will provide a database with fake data for test purposes. First, the test program will make sure that data is properly stored and retrieved from the database, by simulating hundreds of calls to the database interface using valid data. Once the basic functionality has been verified, more calls to the database interface will be tested using bad input, duplicate data, and invalid data requests.

HTML Generator: The testing of this component will focus on the ability to display varying amounts of data. Each template will be tested using data sets of size 1 to N, where N is large. The UI designers, the implementer, and the tester will determine if the HTML generated displays information correctly in an aesthetically pleasing manner. The generated HTML will be viewed using with many various systems (Unix, PC, Mac), web browsers (Internet Explorer, Netscape), and monitor resolutions to ensure that the html generator works properly.

Logical Components: The logical components consist of the user request handler, grade calculation handler, and grade entry handler. A test program will verify the administrative functionality of the components by comparing data sets with contents of the database to ensure that students, classes, and TAs were entered or removed from the system properly. The grade computations performed by the grade calculator will verified by comparing output from the component with predetermined grade calculations.

Access Profile: This component will be tested by simulating operations performed by users with different permissions. The first segment of the testing will make sure that each user can access the necessary information specified by their permission. The second part of the testing will attempt to access information and perform operations outside of the user's jurisdiction. Special attention will be paid to individuals with different roles. For example, the test program will check to see if a student who is also a TA can access grade data from another course using their TA permissions or if a TA can manipulate grades of assignments of which he/she is not responsible.

Security: A random word generator could be used to break into the system by attempting to determine passwords.

Web Server: This component will be tested using stress tests to determine if it can withstand multiple users accessing the server simultaneously and whether or not requests were fulfilled correctly and in a timely manner.

All components should compile without any error or warning messages. In cases where a component is dependent on incomplete components, dummy values will be returned from the incomplete component.

4.2 Integration Testing

Before the integration testing can go (relatively) smoothly, each component should pass its component test. If a component fails some or all of the component test, the bug(s) will be recorded and project members will be made aware of the problem (and expected to fix it ASAP, as it will delay successful integration testing). Integration testing should occur after each component is added to the final system. Partial integration testing will enable the team to track down bugs more efficiently because the problem can be isolated to a few components. Once the system has passed the various partial integration tests final testing can be conducted. If at all possible, all testing should be automatic, because automatic testing is robust than manual testing. However, it is very important that both programmers and testers be on hand and fully informed of the progress of the test programs, as it is easier to find and fix bugs when you observe the misbehavior of the program yourself, rather than hear about it from somebody else.

4.3 User Testing

Initial user testing of the UI designs will occur very early in the schedule. The UI designers will explain the purpose of the software to professors, students, and TAs on a one-on-one basis, and revise their design based on the users' reactions to screen mock-ups and UI flow-of-control descriptions. This is not intended to be a time-intensive process, but rather a second round of requirements gathering which focuses on the users' interaction with the UI.

After integration testing, user tests will be conducted. A group of approximately 10 to 20 students, TAs, and faculty from different departments at Brown will be gathered. The users will be given project documentation and asked to use the program. After using the software, study participants will be asked to evaluate the software using a survey. The following are a general set of questions that might be on the survey:

- Did you find any bugs while using the software?
- Did you have any difficulties using the software? Why?
- Is the documentation easy to follow? Why?
- Is the documentation useful? Why?
- Is the user interface appealing? Why?
- Would you use this software in the future?
- What do you like the most about this software?
- What do you like the least about this software?
- Are there any features that we should add?
- General Suggestion/ Comments

It will be important that each unit be tested thoroughly before integration.

4.4 Security

- Continue multiple attempts to log on as a different user. Be sure that server is never fooled
- Attempt to access grade information for which access should not be granted.

4.5 User Interface

- A number of users must be brought in to test each feature offered on the interfaces. All form entries must be tested thoroughly to ensure that data must not be re-entered if the server runs into an error.

4.6 Grade Calculations

- Use many different grading strategies.
- Try numerous weighting methods
- Set weights to add up to > 100% or < 100%

4.7 Server Performance

- Write an automated testing application which will simulate up to 100 users entering information simultaneously, and up to 300 users viewing information at the same time.
- Push server to see the maximum number of users possible before system crash.

5 Hardware and Operating System Specifications

5.1 System Limitations

- The system must run on a Sun workstations
- The workstation must be running Solaris

5.2 Implementation Limitations

- Code must be written using the C++ language
- All HTML used must be generated by this code.

5.3 Portability

- Design must be portable such that it could be run on a Windows system, for instance.

6 Other Specifications

6.1 Documentation

- A manual must be written which explains the workings and the maintenance needed on the server.
- An extensive on-line help facility must be created for users to access while using the system.
    - On-line help windows must be a separate browsing window to minimize information loss from main window.

6.2 External Dependencies

- The Registrar database is the system's only truly risky external dependency. Because actual integration with the Registrar's system is highly unlikely, the interaction has been abstracted a little bit. All access to and from the Registrar will be in the form of import/export files. This does not preclude future integration with the Registrar. Because the priority levels are so low for this integration, however, the risk is minimized.
- The system is probably going to depend on a Kerberos server for password authentication. The motivation for the façade between the authentication mechanism and the rest of the system is to protect the rest of the system from unnecessary delays should our first attempts at Kerberos authentication prove to be a bust. In the event that Kerberos authentication is unsuccessful, the security person will replace the mechanism behind the façade with a working mechanism (our own password database or file). The motivation for attempting to work with Kerberos, which we feel makes it worth investigating, is the ease of use for the Brown user (one password, many services). If it works, then we won't have to populate a user password database. If it doesn't work, we've minimized the risk by using the façade pattern.
- The use of a third-party SQL/C++ interface, MySQL++, may also be considered an external dependency. MySQL++ was selected based on:
    - The opinions of our peers who have used it, and chose it over PostGreSQL
    - The accolades it has received from "reader's choice" contests and its recognition by Oracle
    - The fact that it has been ported to Windows development environments, which lays the groundwork for a Windows port of our system.

7 Group Organization & Roles

Project Manager – Charles Thompson

This person oversees the project as a whole. This person will manage the project schedule, resolve conflicts, schedule meetings, keep in contact will all members of the team. Will assist in moderate coding.

Architect – Curran Nachbar

This member is in charge of the design of the project. S/he will make the design decisions which will shape the product. Will assist in moderate coding.

UI Designer – Scott Nisenfeld

This individual or individuals have the charge of designing the look and feel of the user interface. The UI designers will also help to design and implement the CGI Generator.

Documentation/Librarian – Joe Wilkicki, Michelle Nguyen

This person is responsible for managing the versions revision tools used, and keeping versions information clean. Also in his/her purview is the responsibility for the documentation for product administration, and for the user help screens. This person will work with the UI designers in creating these screens.

Tool Supporter – Joe Wilkicki

This person will assist in choosing compilers, and libraries, building makefiles, setting up the revisions control software. This person will also help to troubleshoot problems with these tools. This person will also lead in coding the CGI Generator and Session Manager.

Database Designer – Erika Hart

This person is the one in charge of designing, overseeing and managing the databases. S/he will also help to implement DB Controller modules. When database modules are completed, this person will go on to help with coding other modules such as the Web server.

8 Task Breakdown & Schedule

In a separate file, sched.pdf, we have laid out all the tasks involved with making this project work. Included on this schedule are provisions for planning, testing, implementing, integrating, and documentation. Each of these areas is color coded for easier readability. Following is the assignment of tasks as it corresponds to the schedule.

Task	Team Member
Request Dispatcher	Greg
User Request Handler	Hui-Yuan
Grade Entry Handler	Hui-Yuan
Grade Calculation Handler	Changhee
DBI – Database	Erika
Tools/ Web Server	Joe
Session Manager / Kerberos	Greg
Session Class	Joe
Request Parser	Curran
Access Profile Class	Charles
HTML Generator	Eduardo
Template Archive /UI	Scott
Documentation/Help Screens	Joe, Michelle
Testing	Imeh, Changhee

Culture

In order for this type of group to stay a cohesive unit, a number of standards must be in place. These standards ensure that members are communicating effectively, and allow a smooth operation.

Communication
All primary communication will be through email. It is expected that everyone will check the email at least twice per day. The newsgroup may be used to archive information, but no urgent messages will be sent to the newsgroup.

As time is so important in this type of project, it is important that email messages that ask for specific answers must be answered as soon as they are received. If the answer requires more time for thought or investigation, then an acknowledgement message should be sent at that time with an expected time of reply.

Meetings will be kept to a minimum to allow group members to spend their time on their project. On the whole, email will be used for most major communication. We all should expect, however, that there may be one meeting set up each week.

Workload
The enclosed schedule is a tight one. It will require that every team member be committed to getting his/her tasks done. If a member foresees having difficulty in completing a project, he/she must tell the project manager as soon as possible. The schedule is not meant to take over the lives of the member, it is meant to keep the whole group on track in producing an excellent piece of software.

Progress
Each week, by Thursday morning, team members are to send to the project lead a short summary of their progress. This summary must include milestones reached, problems encountered, solutions to those prolems, and any problems foreseen in the next stage of development.

Activities
No group task is complete without the glue that keeps groups together – food and fun. Once per week, the group members will have a gathering in which the objective is to relax. Team members are encouraged to make suggestions about interesting activities in which the group can participate.