**Uni**

Spring 2012

**Full Proposal**

**CS 3300 Final Project**

**Joseph Lim (jcl296), Leonard Lu (lhl38), Zeheng Wang (zw58)**

Table of Contents

[Why Uni? 3](#_Toc321496919)

[User Interaction Cases 4](#_Toc321496920)

[Proposed Architecture 12](#_Toc321496921)

[Application Technology 13](#_Toc321496922)

[Reused Code 14](#_Toc321496923)

[Division of Labor 15](#_Toc321496924)

[Intermediate Prototype Functionality 15](#_Toc321496925)

# Why Uni?

The motivation behind our concept is simple - to leverage on the power of social networks to facilitate course selection and resource sharing among college students, all within the comfort of a single web application. While there are several websites and web forums on the market today that deal with course selection and resource sharing, their functionalities are disparate, and are not powered by social networks. Uni seeks to seamlessly integrate features from existing websites such as Schedulizer and CourseRank, and further add its own value by introducing a dimension of fun and convenience through social interactivity, providing users answers to fundamental questions such as “Which of my friends have previously taken this courses, and what did they think of it?”

Uni allows users to store their list of classes (both current and previous) and ratings for previously taken courses. There is an additional interest in using a Facebook application to achieve this as one is more likely to hold a friend’s opinion in higher regard than reviews from strangers in an open forum. Additional features include allowing users to create quizzes and attempt quizzes created by others in the class. Friendly competition is promoted through the creation of a leader-board system which rewards users for meaningful actions.

The real value of the app, however, arises from its ability to recommend courses in a non-trivial fashion to users. We will build a recommendation algorithm based on the user’s optional preferences, and course ratings (with friend ratings carrying a heavier weight). This is especially useful in scenarios where users have a broad range of possible courses to select from but want to make a quick decision about which course to take, rather than having to sieve through individual course catalogues and reviews. Therefore, the application will be most effective once it has accumulated a significant user base.

# User Interaction Cases

The following user interaction cases will provide a clearer picture of the functionality of our application. These use cases will cover all the fundamental interactions a user might have with our application. We will clearly illustrate each case with an activity workflow diagram and/or a mock-up screenshot. For purposes of brevity, please note that these cases may not include the optional add-on functionality we may develop later on in the app. Also, the mock-up screenshots are purely for illustration purposes and do not reflect the final design of the app.

1. **Gaining Permissions and setting up**  
   Like all Facebook applications, Uni will prompt the user for permission to access their personal information. Gaining access to user’s personal information is key because Uni requires information on the user’s name, friends, profile picture, etc as we will discover in the later use cases.An additional security measure we hope to explore is to cluster people by network, in other words, those who have not registered their cornell.edu email account to be part of the Cornell network will not gain access to Cornell-related courses. This will be trivial at the outset, as we will only be implementing Cornell courses but will provide a means to scale in the future.  
     
   Upon approving Uni to gain permissions, the user will be prompted to key in two pieces of additional information – 1) major, and 2) year of study.

We are now ready to begin using the app! The main, introduction page of the application features 3 links:   
1) Find Courses,   
2) Find Friends, and   
3) My Profile.   
  
We decided to house all our functionality within these 3 pages for simplicity and user-friendliness, as Facebook users are often abhorrent of UI complexities.

1. **Find, View, Add, and Rate Courses**
   1. Find Courses

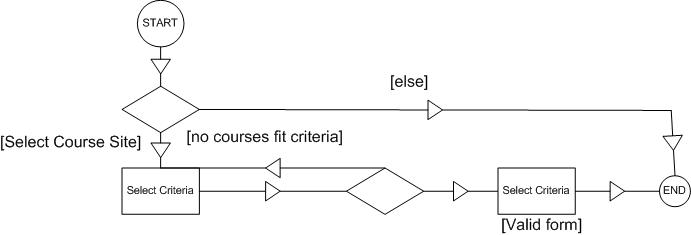
Clicking the Find course link brings us to two sub-choices, ‘By Class Number’ or ‘Recommended Classes’–

**I) By Class Number**A user selects this option if he knows the exact course he wishes to search for (i.e. CS 3300).

(i) User selects ‘By Class Number’ option

(ii) A form appears, which the user fills in and clicks submit.  
(iii) If the form has been filled out correctly, a list of search results appears. If no results, prompts the user to search again or terminate. A complete form is simply having at least the department name and no malformed input. A specific course number is not mandatory in every search.  
(iv) User may now choose to click on any of the search results to navigate to the course page



****

**II) Recommended Classes:**A user might select this option if he wants the system to recommend him a class. For instance, imagine a user who wants to search for Econ or Psych courses above 2000 Level, of 3 credits or more. He does not know which specific class to select, but wants a recommendation based on his preferences (and what his friends may be recommending).  
  
(i) User selects option ‘By Recommended Classes’  
(ii) A form appears, with a list of optional criteria that user may choose to fill out (i.e. leave empty if no preference). The criteria is a combination of hard preferences (only 3 credit classes, only Econ department, etc) and soft criteria (less than 100 students in course, etc.)

(iiI) A list of recommended results appear, filtered by the user’s criteria, and ranked according to friend’s recommendations. If the form was left empty, the user simply sees a list of all courses listed in order of friend ratings.

(iv) User may now click on any of the search results to navigate to the course page.

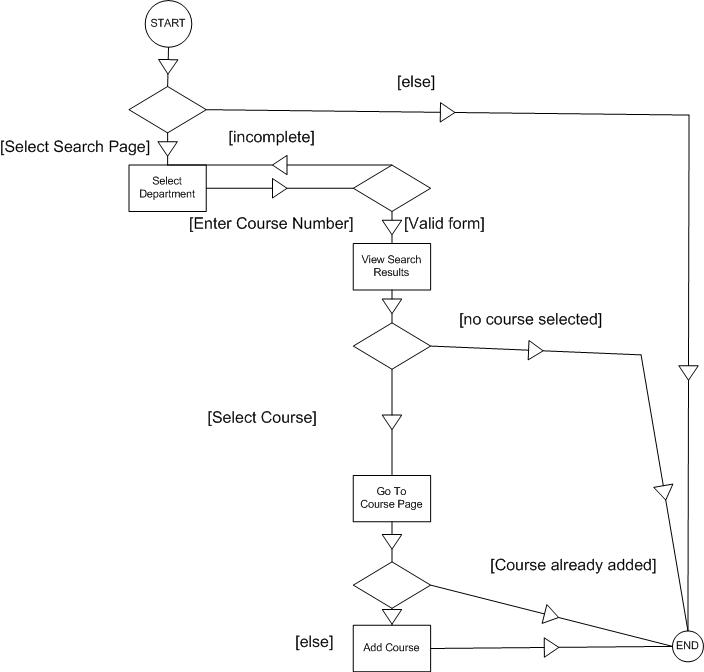
****  
**\* Note: Criteria given in screenshot is for illustrative purposes and not exhaustive**

* 1. **View Courses**Upon clicking the link to a course the user will be brought to the course page, where he can review information on the course (description, syllabus, average friend rating, etc). The course link can be presented after searching for a course, viewing your profile and looking at your enrolled courses, or looking at other people’s profiles and seeing their enrolled courses.

# Macintosh HD:Users:leonardlu:Desktop:courses page.png

* 1. **Add course**

On every class page, there is a button to add (or drop if already enrolled) in the class.

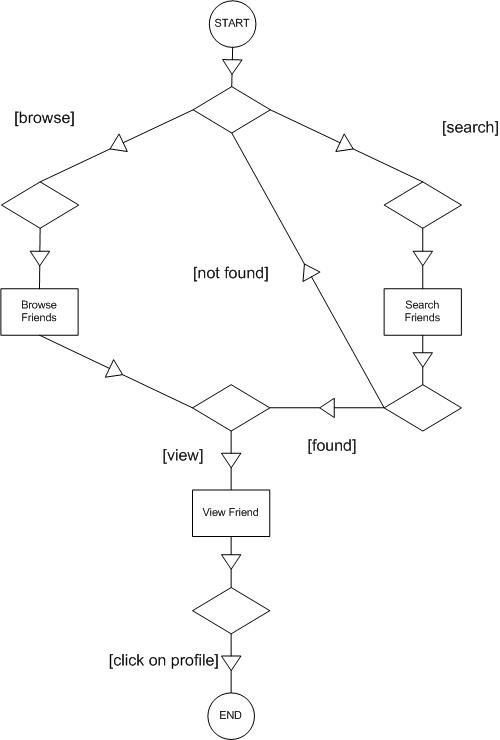


* 1. **Rate Course**

From the course page, there will be a numeric rating system that will depend on user input, which will then be stored in the database for others to see and to subsequently generate recommended courses for others. The user will rate and submit, and will allowed to re-rate and resubmit as desired.

1. **Find Friends**Clicking the ‘Find Friends’ link from the main page brings us to the following page with a list of the user’s friends who are using the app, and the courses they are enrolled in

# Macintosh HD:Users:leonardlu:Desktop:friends page.png

The user can search for a specific friend as well. Clicking on a friend’s name brings us to the friend’s profile summary.  
  
This is illustrated in the activity workflow diagram below:  


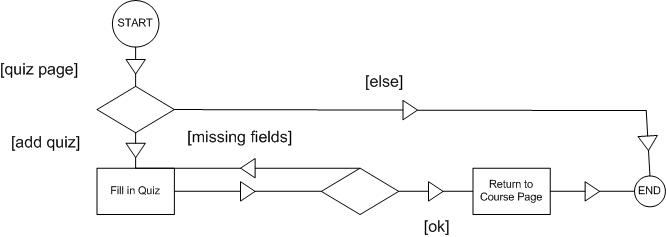
1. **Quizzes**

A key function of our application is the ability to make and take quizzes. This provides a level of social interactivity where classmates are able to help one another out with course material before examinations, as well as provide a means of reaching out to quiz makers with questions. The following tasks will assume that the user is starting from the course page. From that point, one would see a section labeled “Quizzes” that will provide a list of quizzes ordered by creation and whether one has completed it.

1. **Make Quiz**

The quizzes section will have a button to “Add New Quiz” that will provide a premade form that provides a further option to add more than the predetermined number of questions. Each question will have a form for the question, four potential answer choices, with radio buttons to decide which answer is correct, and an optional textbox for further explanation.

Once the user completes the form, s/he will submit it and will be directed back to the Quizzes page.



1. **Take Quiz**

From the Quizzes page there will be a list of quizzes for the course that was chosen. The user simply needs to choose which quiz to take, take the quiz, and submit. The results will be calculated and the quiz redisplayed with the correct answers shown.

# C:\Users\Zeheng\Desktop\Spring 2012\INFO3300\Final Project\DoQuiz.jpg

# Proposed Architecture

**General Description**

We implement a top-down approach in the design of our proposed architecture. We first begin with identifying the high level three-tier architecture:

* Presentation Layer: Responsible for rendering the User Interface (UI)
* Business Tier: Responsible for processing business logic
* Data Tier: Responsible for interacting with the data storage system

The importance of the three-tier architecture is the linearity of the architecture – i.e. the presentation layer should never interact directly with the data tier, and vice versa without passing through the business tier.

However, because of the overlap between the three tiers, it is inappropriate for us to directly classify each tier as a sub-system. To identify our sub-systems, we have to more closely analyze our application framework.

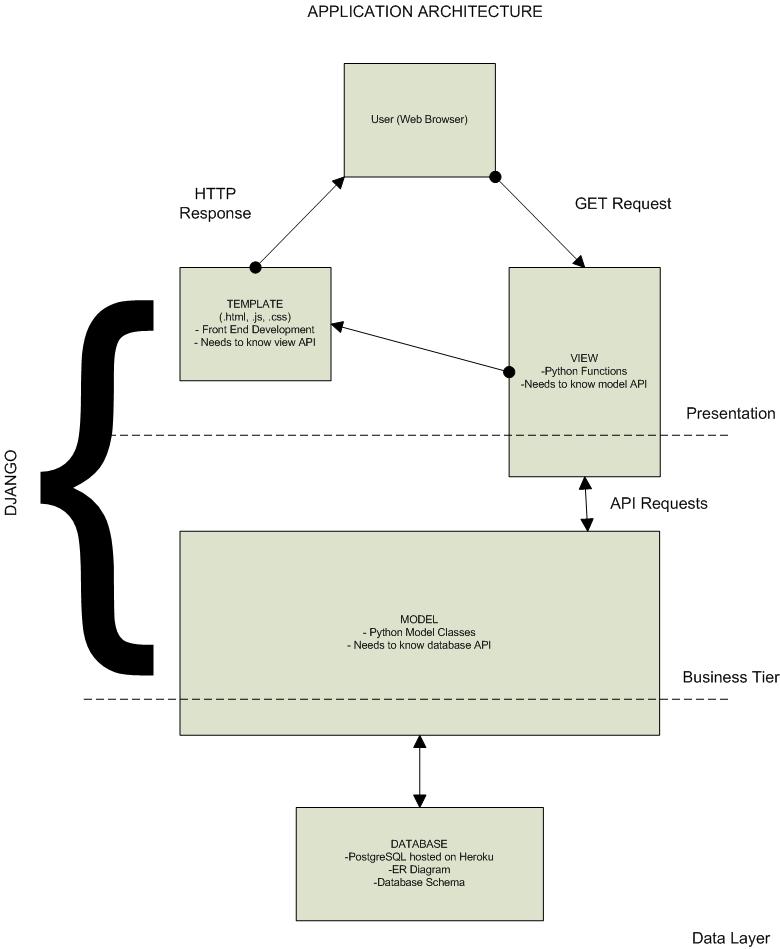
Because Django is our chosen development framework, it makes sense to use the Model-Template-View (MTV) framework inherent in Django itself. It is crucial for us to realize the critical difference between MTV and Model-View-Controller (MVC) descriptions to avoid confusion from here on – In MTV, the ‘template’ deals with data presentation while the ‘view’ deal with what data is being presented, but in MVC, the ‘view’ deals solely with the presentation (i.e. ‘template’ in MTV), while the ‘controller’ is the one choosing which data to present). From here on, our definitions of ‘view’ will strictly follow the MTV description, as follows:

* *Model:* This layer contains anything and everything about the data: how to access it, how to validate it, which behaviors it has, and the relationships between the data.
* *Template:* This layer contains presentation-related decisions: how something should be displayed on a Web page or other type of document.
* View: This layer contains the logic that access the model and defers to the appropriate template(s). You can think of it as the bridge between models and templates.

*(From* [*http://www.djangobook.com/en/2.0/chapter05/#cn16*](http://www.djangobook.com/en/2.0/chapter05/#cn16)*)*

As illustrated in the architecture diagram below, we find that ‘View’ overlaps both the presentation layer and business tier, while ‘Model’ overlaps both the business and data tiers. While considering the (numerical) strength of our team of 3, it becomes appropriate to identify 3 subsystems, as follows:

1) Template  
2) View  
3) Model and underlying database



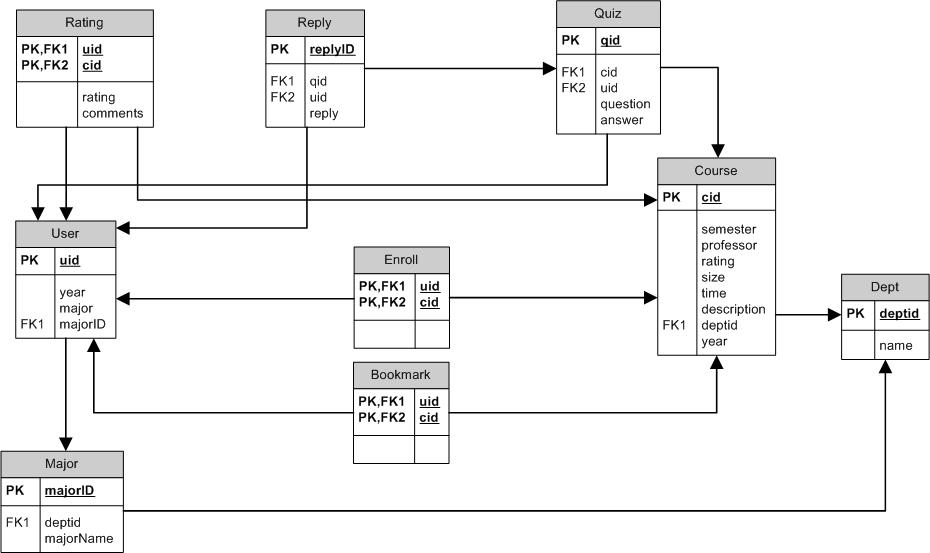
A clearer representation of the subsystems is represented in the relationship graph below:



**Data Model in each subsystem**

1) Model and underlying database

Database Schema  
Our database schema (tables and dependency) is concisely summarized in the ER diagram below:



Model

Our data model for Model can be simply abstracted to four classes:

2) Middle Tier (View)

3) Template (Client)

We will set up four classes representing each ‘type’ of page the user will view, i.e. Search, Course, Profile, and Recommended. Each class has its associated

**API**

Importantly, we want to build an API that is RESTful, i.e. in accordance with the six constraints below:

* Client-server separation (in accordance with separation of concerns principle)
* Stateless (no client context is stored on the server between requests)
* Cacheable
* Layered system
* Code on demand
* Uniform interface

Please refer to Appendix A for API. The different subsystems may be required to know other subsystems’ API, as outlined in the Application Architecture diagram and relationship graph above.

**CRC Cards**

CRC (Class-Responsibility-Collaboration) Cards are an extremely useful way to keep track of one’s tasks and important considerations of the task. It gives information about what the class does (responsibility) and the other classes this class interfaces with (collaboration). In other words, it is a plain English description of our API, with responsibilities representing class methods and collaboration signifying dependency.

Refer to Appendix B for detailed CRC cards for both the template and view.

# Application Technology

**Back End**

App Hosting on Heroku  
We will be deploying a Python/Django application to Heroku using a PostgreSQL Database as the backend. Due to the complexity of the information stored in the database, it is necessary to use a relational database and Postgres seems to be the best option since support for MySQL is not provided.  
  
Possible File Hosting on Amazon S3  
As a potential add-on to our application, the course forum will allow users to upload files to promote resource-sharing. If this functionality takes off, Amazon S3 could provide a solution to trying to host files on Heroku  
  
**Middle Tier**

We will utilize Django and Python as our web development framework because of the team’s familiarity with the language, in opposition to Ruby on Rails. Django provides a clear-cut separation between ‘model’, ‘template’, and ‘view’, which we have already illustrated in our architecture description.  
  
**Front End**

Traditional HTML, CSS and JS will drive front-end development, with Pyjamas being used to compile Python to Javascript. We will also occasionally reference the Facebook API (FBHTML, FBJS, FBSQL) for Facebook-related functionality.

**Code Sharing and Version Control**

For the purposes of viewing and modifying code collaboratively, we will be using the Git distributed version control system. Importantly, Git will enable us to keep track of changes through logging, as well as create experimental branches for adding layers of functionality, without compromising the existing code.

**Concurrency Control**

We expect inherent concurrency controls in the PostgreSQL database system (MVCC) to take care of data consistency issues in the database. It is therefore also vital that we transmit error messages between each layer (from the database layer to the model layer and then to the client interface level) in the event where concurrency issues prevent operations from executing.

# Reused Code

Since this application is a entirely novel, it will be developed entirely from scratch.

# Division of Labor

Each member of the team will be given a sub-system which he will take complete ownership over. In the case of our application, Owners will partake in collaboration with neighbors as illustrated in the relationship diagram above.

|  |  |
| --- | --- |
| Team Member | Responsibility |
| Leonard | Model + Database |
| Zeheng | Template |
| Joseph | View |

Workflow is illustrated in the timeline below:

# Intermediate Prototype Functionality

Because the integration with Facebook is critical for our application to have any meaning, our intermediate prototype will feature a fully integrated Facebook application, but with bare-bones functionality, as laid out below:  
  
1) User is able to search for, and add classes from a pre-populated database  
2) User is able to view his or her profile and drop classes from the profile  
3) User is able to enter a course page and enrol in the course  
4) User is able to view a list of friends and the courses they are enrolled in.   
5) All of the significant user actions are reflected on the user’s Facebook wall.  
6) Any user action that prompts an interaction with a friend (for example, adding a course that a friend is enrolled in, or doing a quiz posted by a friend), results in a notification in the friend’s application  
  
**Functionality to be added in final product (in order of priority of implementation)**  
  
1) Build a recommendation algorithm to rank recommended courses, with optional filters if desired by the user  
2) In the course page, user can either create a quiz or attempt an existing quiz  
3) Forum page for every class page  
4) Introduce a leaderboard by awarding points for every meaningful action (adding a quiz, completing a course rating etc) to increase social interactivity of application

**Appendix A: API**

**I. Models API (this is the structure in the models.py file)**

**class User:**

int userId

unique userId

String year

school year of user (Freshman, Sophomore, Junior, Senior)

String majorId

unique majorId

**class Course:**

int courseId

unique courseId

String semester

Semester this course is offered (Spring or Fall)

String year

Year that this course is offered

Double averageRating

Average Rating for this course

int classSize

Class Size for this course

String time

timeslot for this course

String courseDescription

course description/syllabus of course

int department Id

department Id of this course

**class Rating:**

int userId

user Id of user that created this rating

int courseId

course Id of the course that the user rated in this rating

int rating

rating given by the user

String comments

comments associated with this rating

**class Quiz:**

int quizID

unique quiz Id

int userId

unique user Id

int courseId

unique course Id

String question

quiz question

String answer

answer to the question, as posted by the creator of quiz

**II. Middle Tier API (views.py)**

User Functions

enrollClass(int courseId)

Enrolls this user in the course

dropClass(int courseId)

Drops this user from the course

takeQuiz(int quizId)

Takes Quiz

makeQuiz()

Makes a New quiz

replyToQuiz(int quizId)

Replies to this quiz

rateCourse(int courseId)

Rates this course

bookmarkCourse(int coursed)

Bookmark course

getCourses()

Gets a list of courses this user is taking

Course Functions

getListOfUsersEnrolled()

get array of user Ids enrolled in this class

**III. Client API**

**class Search:**

loadResults()

Display searched courses

takeUserQuery()

Receive user input and give to middle tier

bookmarkCourse()

Flag course as bookmarked

loadCourseInformation()

Display course information from backend

Sort Results()

Input loaded results and search by criteria

**class Course:**

rateCourse()

Display course rating

addCourse()

Add course

loadCourseDescription()

Display course description

loadComments()

Display course rating comments

takeComments()

Add rating comments

loadAverageRating()

Calculate and display average rating for course

loadFriendsInClass()

Display friends in course

loadQuizzes

Display Quiz List

**class Profile:**

loadUserProfile()

Display user information such as name, year, major

loadCourses()

Display user’s enrolled courses

dropCourses()

Drop enrolled course

loadBookmarkedCourses()

For courses that are bookmarked, display additional indicator

**class Recommended:**

takeUserFilter()

Receive input criteria and package them for use

loadResults()

Display returned results

sortResults()

Algorithm to sort courses by friend recommendations and ratings

bookmarkCourse()

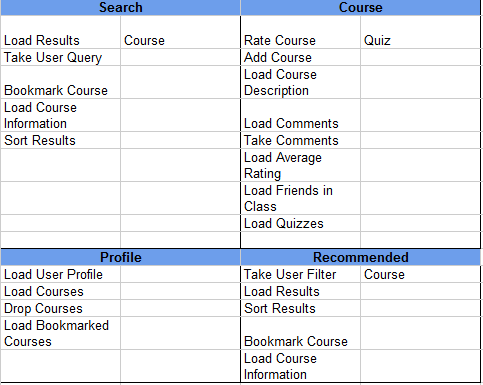
Mark course as bookmarked

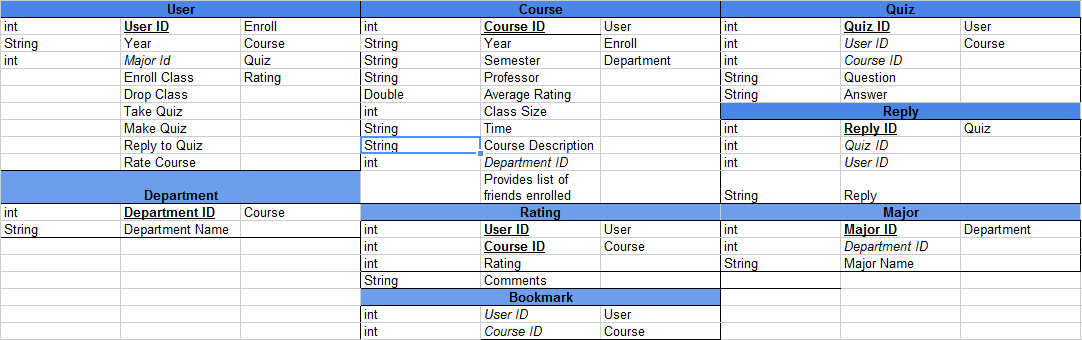
loadCourseInformation()

Display course information

**Appendix B: CRC Cards**

**Front-end (Template)**



Middle-tier (View)