OfCourse

Ziling Zhao, Aron Hershberger, Cosmin Barsan, Nick Brekhus, Ryan Timmons, Cedar Bristol

System Design Specification and Planning Document

Draft 1
4/22/2008

CSE 403 - CSRocks Inc.

OfCourse
Ziling Zhao, Aron Hershberger, Cosmin Barsan, Nick Brekhus, Ryan Timmons, Cedar Bristol

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	All	First Draft	104/22/08

OfCourse
Ziling Zhao, Aron Hershberger, Cosmin Barsan, Nick Brekhus, Ryan Timmons, Cedar Bristol

System Architecture

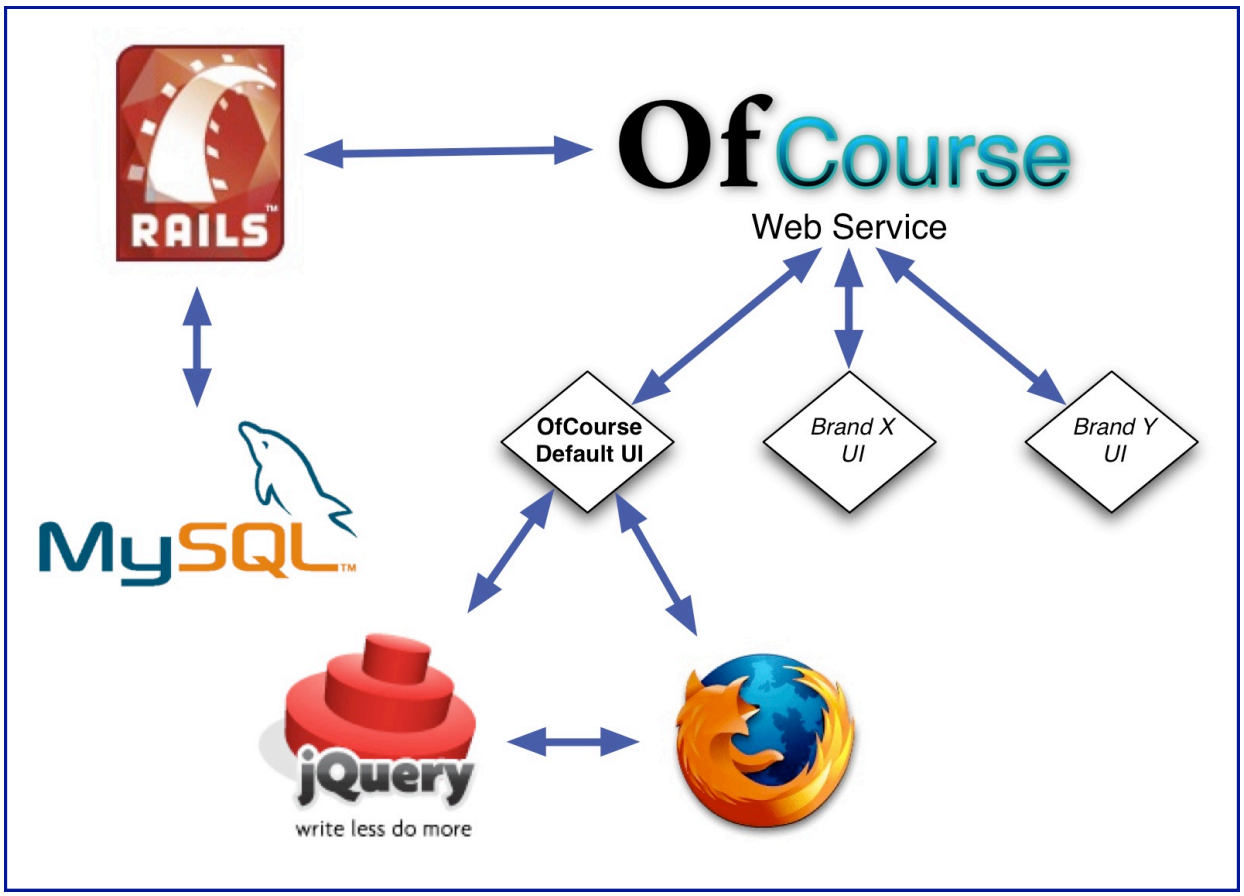
Introduction

OfCourse provides a relational view of courses and instructors aggregated by user-selected criteria. Input about courses is gathered from the UW course catalog and is merged against similar data gathered about instructors. Users are then asked to rate courses and instructors based on an arbitrary set of metrics and are also asked to provide free-form text feedback. Courses and instructors can then be browsed in order of arbitrary weightings of the metrics. Other information about courses is also included such as official pre-requisites, peer-recommended pre-requisites, and peer-recommended alternatives.

The purposes of our project is to be more of a web service than an end-point complete with UI: we are designing a full external API for other web servers to connect and gather our data. We are, however, also developing our own UI to interface with the service. This not only diversifies the project but also ensures we build in an appropriate API ("dogfooding" the API, if you will). Our UI will be composed of the standard (X)HTML, CSS, and JavaScript layers but will also incorporate the jQuery JavaScript framework for AJAX and dynamic DOM manipulation. The fact that our entire view/controller interaction is a web service turns out to be beneficial to us as well when we develop our own UI for the service as it means that absolutely every system call to view/update data can be made via API calls with, e.g., AJAX.

System Architecture View

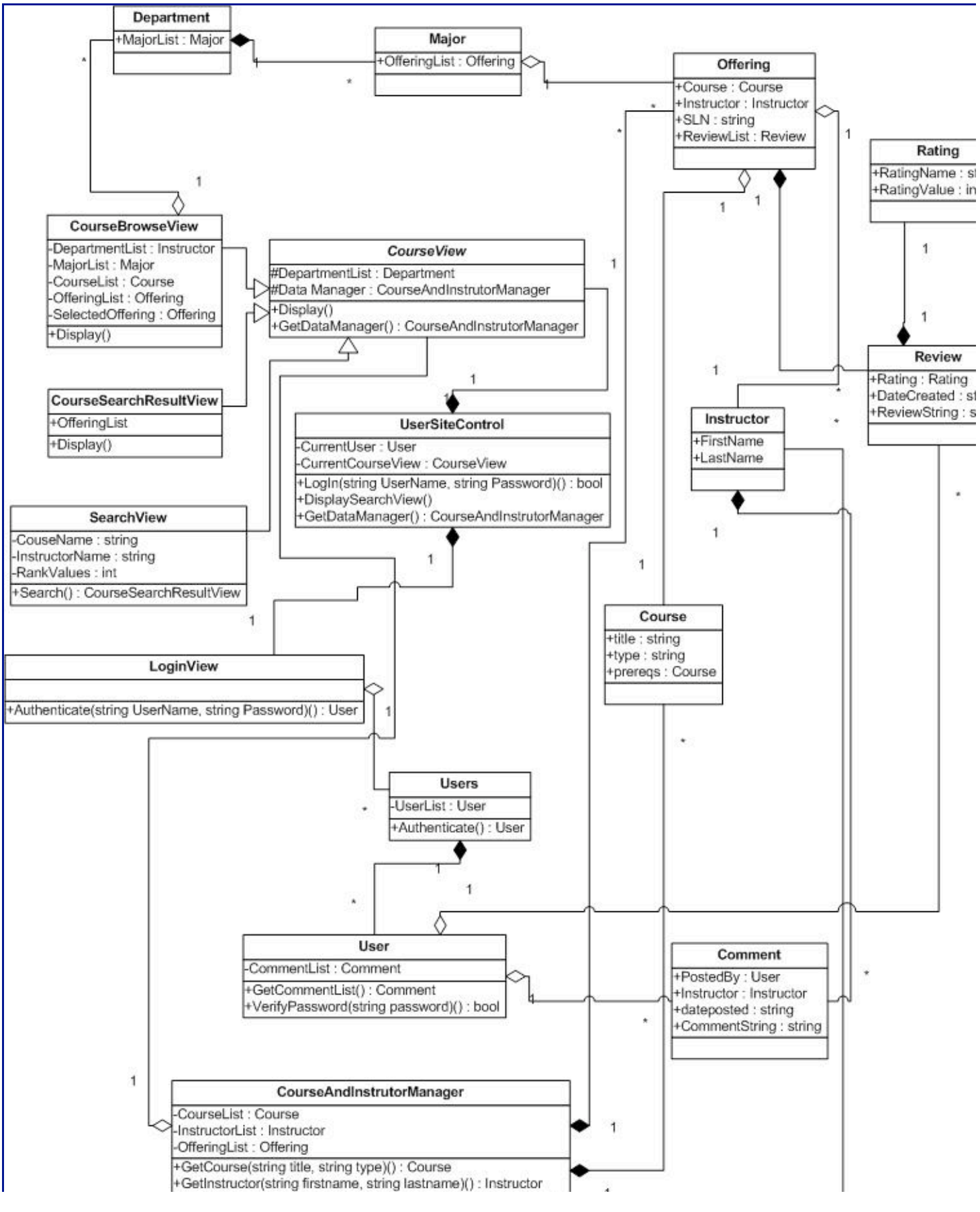
We are building this system using the excellent Ruby on Rails framework to coordinate between the database and the front end. To this end, Rails serves as an object-relational mapper (ORM) connecting to (either Postgres or MySQL depending on deployment) SQL database server. Rails is for most purposes completely abstracted from database implementation, so our project will load just fine on PostgreSQL, MySQL, and even SQLite.



Design view - UML class diagram

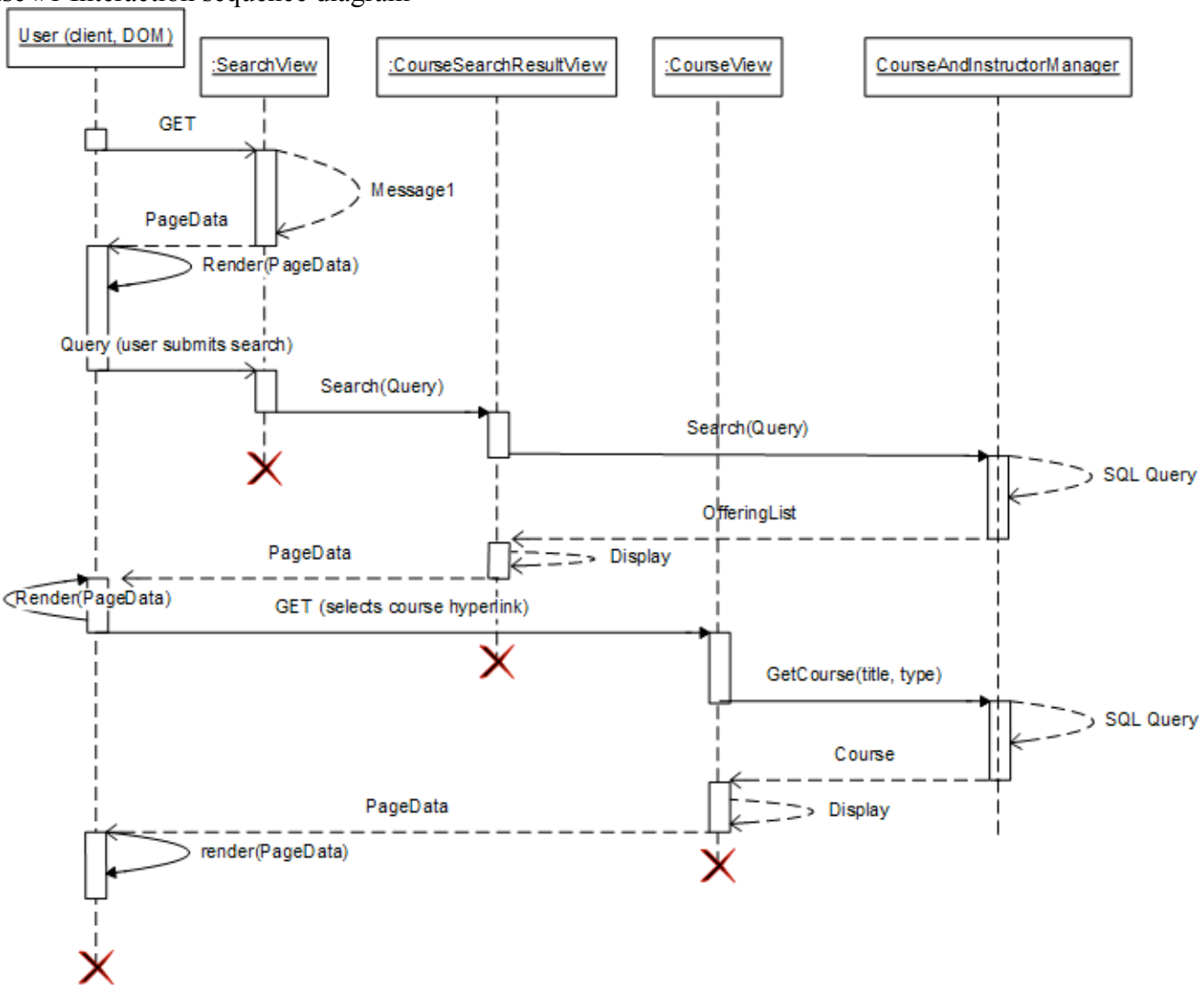
OfCourse

Ziling Zhao, Aron Hershberger, Cosmin Barsan, Nick Brekhus, Ryan Timmons, Cedar Bristol



Process view

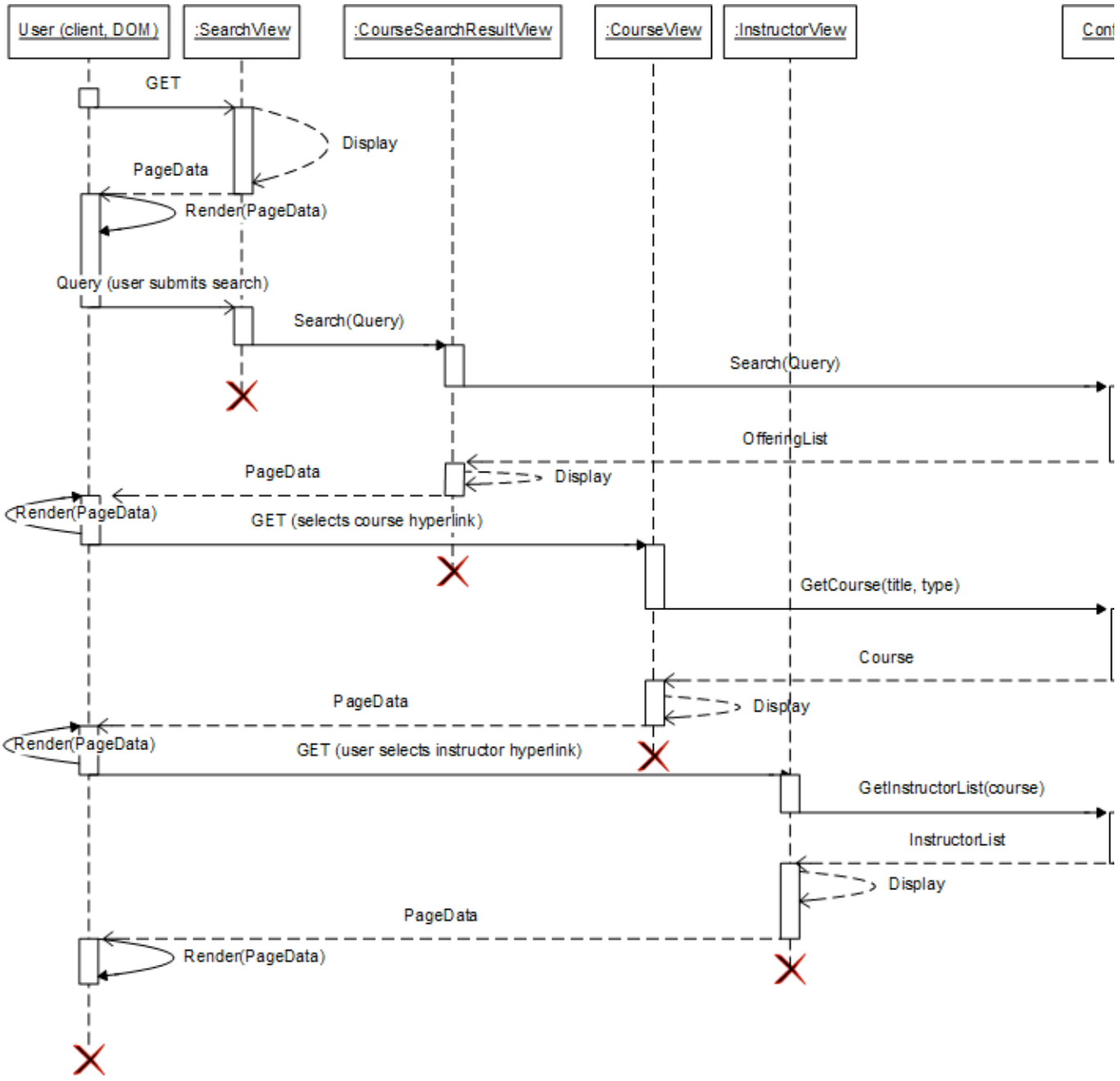
Use Case #1 Interaction sequence diagram



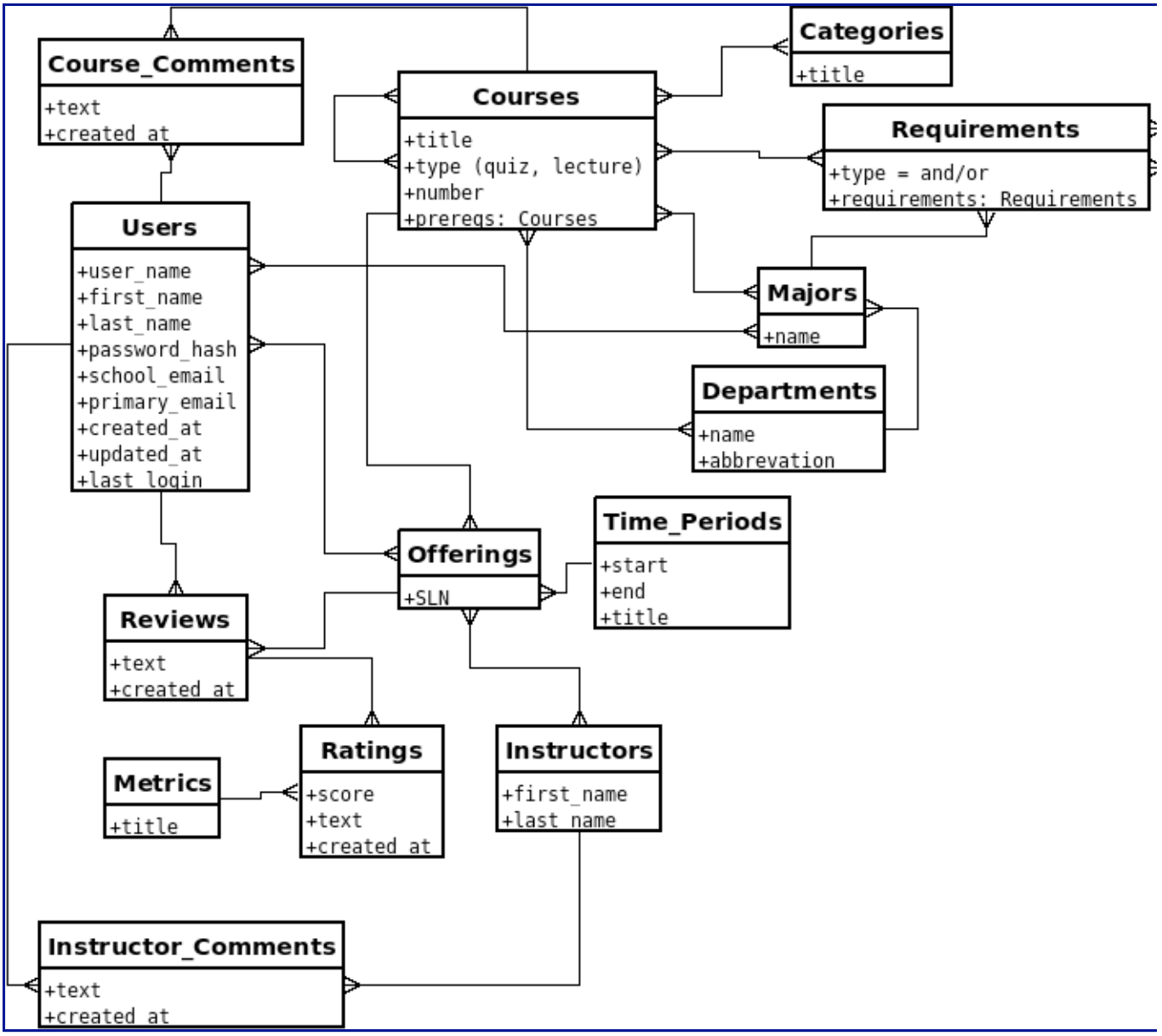
4.

Use Case #2 Sequence Diagram

5.



Database Schema

6. 

Design Alternatives and/or Assumptions

We are assuming that the best approach (initially) for our target audience is through the construction of a browser-based "rich" web application. Since we are offering our data primarily as a web service to other potential front-end providers, our own UI is not strictly necessary and is only one of the many possible interfaces. Alternatives to this include an Adobe AIR application or a Facebook application. Our assumption, however, is that the web application will offer an easily accessible way to interact with our data without being restricted by the interface devised by Facebook. Adobe AIR requires the user to download extra software in order to use the application.

We also assume a certain amount of tech-savvy in our target audience. We do not plan to build a large support or "help" section for the site but instead plan to use common user interface elements and to do a

certain amount of usability study. The assumption here is that most users of the site are college students. Given the popularity of sites such as Facebook, whose target audience matches ours, it is safe to make this assumption.

Development Plan

Team Structure

- **Cosmin Barsan** - Scheduling/turnin, Integration Lead, Testing
  - **Scheduling/turnin -** Tracking what items need to be turned in when and insuring everything is turned in on time.
  - **Integration Lead** - Verify that all components work correctly together.
  - **Testing -** Assisting the Testing lead in verifying all components work correctly individually.
- **Nick Brekhus** - Testing Lead, Usability
  - **Testing Lead -** Provide oversight and assist in the implementation of our testing strategy.
  - **Usability** - Verify that site meets usability expectations as part of our testing strategy.
- **Cedar Bristol** - Model Lead, Controller Lead
  - **Model Lead** - Implement the data model. Make sure that there are tables to put data in and methods to update data and sample pages to show other developers how to call them so that they can get do their parts. Collect feedback from other developers on the model components.
  - **Controller Lead** - Ensure smooth communication between model and view components.
- **Aron Hershberger -** Documentation Lead, Data Migration Lead, Customer Relations
  - **Documentation Lead -** Insure all documentation (including online User Help and Admin Help/API documents) gets written and is clear and readable, final review and editing/formatting of all documents submitted to CSRocks to insure they are complete and free of errors.
  - **Data Migration Lead** - Write a parser and scrape data from publicly available sources to initially populate the database.
  - **Customer Relations** - Coordinate and participate in all customer/vendor interactions, providing a single contact point for our customer/vendor.
- **Ryan Timmons** - Project Lead, UI Lead
  - **Project Lead**: Coordinate the "direction" of the project: are our features showing up in our code and in our product? Are we suffering from feature-creep or are we behind on deadlines? Is the team sure of what it's doing at any given time and why it's doing what it's doing? The project lead, along with being super wicked-awesome, answers these questions. And eats cake.
  - **UI Lead**: The UI lead is the point-contact for what is being worked on and what is the current status of development of the user interface. The UI lead ensures that aspects of the UI are consistent with the project's goals and feature set and that components of the interface are done at the right times and on-schedule.
- **Ziling Zhao** - API Lead, Version Control, Internal Systems Management (trac)
  - **API Lead** - ensures that the necessary data is available external to the application and that the application is able to receive data from external sources. The API lead must also work closely with the Model lead and the Integration lead to ensure that all necessary components of the

architecture are accounted for whenever data is transfered among application layers or external calls.

- **Version Control** - Verify that changes submitted meet coding standards, makes sure that people are keeping with proper version control etiquette, and overseeing merge and branch operations.
- **Internal Systems Management** - Keep trac, svn, and reference application up and running. The reference application being the trunk version of our application.

Team Communication

- **Meeting Times** (meetings twice weekly, all group members should try to get to each meeting and **must** come to at least one meeting per week)
  - Tuesdays 7:30pm CSE 006
  - Thurdsays 10:30am CSE 006
- **Wiki** at https://www.enragedcoconut.net/trac/OfCourse includes the minutes from each meeting.
- **Mailing list**: cse403-ofcourse@u.washington.edu - all email from the list should be read and (if appropriate) responded to within 24 hours.
- **Individual and group progress**, tasks, bugs, etc. are tracked by tickets in our **Trac** instance online at https://www.enragedcoconut.net/trac/OfCourse/report, which all group members must check and update regularly.

Project Schedule

Task/Milestone	***[Estimated effort]***	***Date due***	***Resource(s)***
Set up trac	1 Days	April 9	Ziling
Schema	3 Days	April 21	Cedar, Ryan
Scaffolding	3 Days	April 22	Cedar, Ryan
UI Prototype	2 Days	April 30	Ryan
Zero Feature Release		April 30	
Unit Test Framework	3 Days	May 4	Nick

Finish parser and import parsed data into database	5 Days	May 7	Aron
Functional Test Framework	3 Days	May 7	Nick
Basic View / Search Functionality	4 Days	May 9	Ryan
Integration Test Framework	3 Days	May 10	Nick, Cosmin
Basic Edit Functionality	4 Days	May 11	Ryan, Cedar, Zilling
Beta Release		May 12	
Browse UI funtionality	5 Days	May 16	Ryan, Cedar, Zilling
Beta2 Release (for customer eval)		May 19	
Advanced Search	3 Days	May 22	Ryan, Cedar, Aron
User Accounts Feature implemented	4 Days	May 30	Ziling
Usability Verification	1 Day	June 1	Nick
Online User Help documentation complete	2 Days	June 2	Aron, Nick
Online Admin/API documentation complete	2 Days	June 4	Aron, Ryan
Final Release		June 5	

Risk Assessment

Risk	**Chance of occurring (High, Med, Low)**	**Impact if it occurs (H,M,L)**	**Steps taking to in crease chance it won't occur**	**Mitigation plan should it occur**
Too UW specific (inflexibility of architechture)	High	Low (planned limitation)	Code the project as generically as we can without sacrificing ease of use and implementation. Deciding we don't really care very much if this risk occurs.	Trying to expand the project beyond UW would be at best a stretch feature at this point, so we won't really care.
Subject to "the network effect"	High	Med	Advertise	Advertise More
Abuse by Malicious users	Med	Med	Rudimentary Authentication procedures	Provide a well documented and easy to use process for administrators to maintain the data.
Difficulties with Data aggregation	Med	Low	Write a good parser to seed the database	Encourage users to correct the data.
Difficulties with data representation (espeically prerequisites)	Med	Low	Designing alternate interfaces.	Represent Prerequistes textually instead of in a complex boolean structure.

OfCourse
Ziling Zhao, Aron Hershberger, Cosmin Barsan, Nick Brekhus, Ryan Timmons, Cedar Bristol

Test and Documentation Plan

Test Plan - Ziling

1. **Unit Test Strategy**
    1. The objective of these tests are to ensure that the individual components work as advertised.
    2. Every single component of our system is to include unit tests. Our development strategy is test driven, before any component is to implemented, a test and spec must be written and reviewed by team members.
    3. These tests will be run as part of the system test, on commit basis as well as by the developer. It is the commiter's responsibility to test as he implements.
2. **System Test Strategy**
    1. This tests the system on a larger scale, on how the components integrate together and how well they perform their tasks and operate within scenarios.
    2. These tests will involve both user interaction as well as a large amount of interaction. Since our software is primarily a service, it would be trivial to implement automated scripts that would test the system on a daily basis. A simple test framework would be constructed to hook into subversion commits in order to test to see if they have any regressions.
    3. Tests will be conducted in a large scale every day/week, with small tests being run on each commit to trunk.
3. **Usability Test Strategy**
    1. We will test to make sure that the interface is usable. Usability is defined as the ease and efficiency of which people can utilize our tool in order to complete a task or set of tasks. This would both involve utilization of the tool when it is foreign (e.g.: learning curve), as well as methods of acceleration as familiarity with the tool increases.
    2. These tests would mostly involve review of the site using design heuristics such Nielsen's Top Ten as well as past experiences and knowledge of our team members. In addition to this, usability tests may be conducted on subjects outside of our group.
    3. Design testing will be run as often as user interface changes are made. External/customer testing will happen much rarely due to constraints on resources as well as time.
4. **Adequacy of Test Strategy**
    1. We expect the strategies described above to be adequate. Since our development will be test driven and the system will be tested frequently both in terms of components and as a whole, we expect the final product to function as the user intends it.
    2. In order to ensure that our test strategies remain adequate and system quality is monitored, we have appointed a test lead to oversee the implementation of these strategies.
5. **Bug Tracking and Ticketing**
    1. We are using trac to track bugs (as well as tasks and etc.). See https://www.enragedcoconut.net/trac/OfCourse/report/6 .

Documentation Plan

Online User guide ("help files", how to use) - This will include online help links or popup text for most features visible to the users, as well as a general "User Guide" and FAQ.

Online Admin Guide (how to set up) - A set of pages clearly explaining how to set up and administer the application.

Developer guide (api documentation) - Pages explaining the rails objects and the web service api.