CourseOwl

A recommendation system to track and discover
online courses from multiple providers.

Project Documentation

Michael Staszel
Erik Chen
Zheng Kang
Daniel Garcia-Carrillo
Akshay Singh
David Eisenberg

Table of Contents

Project Information

Description

With the abundance of possible classes that one can take online, it is highly unlikely a student will ever get to see or take some of the ones that are most interesting to them. CourseOwl is a system that will provide recommendations to people based on their interests and courses they have taken. The user will be asked what interests they have and will have the option to select subjects already known or learned. They will then receive recommendations of classes they may like from MOOC providers such as Coursera, Udacity, edX, and iversity.

Development Process

The CourseOwl team followed the Scrum development process. We decided to follow this process after following XP in CS 427 for a number of reasons. Scrum and XP are both agile processes, unlike more rigid processes such as Waterfall. Scrum allows customers to change requirements at any time, re-prioritizing as necessary. Scrum follows XP's concept of iterations with the concept of "sprints". Sprints are two-week periods during which developers implement all features. At the beginning of each sprint, developers have a discussion about the tasks the customer would like implemented. At the end of each sprint, developers discuss what they accomplished in the sprint, what issues they had, and what they need to focus on in the next sprint.

Scrum consists of three roles: product owner, development team, and Scrum Master.

The product owner on the CourseOwl was the whole team. At the beginning of the project, Erik and Michael came up with loose ideas for what CourseOwl should do as well as some user stories.

The Scrum Master on the team was Michael, who led the Scrum meetings at the beginning of each sprint as well as the "daily" Scrum meetings that we had twice a week.

After the Scrum meeting, developers were left to implement their user stories. In order to keep track of changes, our team used GitHub and its issue tracker. Our team used GitHub's pull request and branch system to manage changes in the code.

First, developers choose a GitHub issue to work on, and then assign it to themselves. Then, they create a branch using Git, our version control system, implement all code and tests necessary, and push their branch to GitHub. Then, they create a pull request, which is reviewed by another developer. The reviewer can then comment on lines of code, the commit, and see whether Continuous Integration tests pass. Then, they can either accept the change or deny it.

Our team used a similar process in our refactoring. Developers relied on pull requests, branches, and our Continuous Integration process. Every time a developer made a change to the code, our test suite would run on a server and automatically email the team, as well as update GitHub. In case tests failed, GitHub would show warnings when reviewers looked at the code.

For some user stories, developers pair programmed. This was not required, as it was for XP. Developers could pair on whatever user stories they wanted. Developers pair programmed on "large" user stories, like the recommendation engine, as well as when they needed help, like at the beginning of the semester, when some developers were not familiar with the languages and frameworks we were using.

Requirements & Specifications

User Stories

As part of our CourseOwl project, our main goals were broken up into "user stories," short sentences of what actors might want to do using CourseOwl. Over the course of the project, user stories were revised, rescheduled, added and removed. This is our final list of user stories implemented:

1. As a user, I want to be able to see a nice-looking homepage.
2. As a user, I want to be able to log in using Google and Facebook.
3. As a user, I want to see some information about this project on its homepage.
4. As a user, I want to see courses offered by (at least) one online course provider after logging in.
5. As a user, I want to be asked what courses are interesting to me as part of the sign up process.
6. As a user, I want to be able to see courses from multiple (at least 2) providers on CourseOwl.
7. As a user, I want to mark which courses I have taken in the profile page.
8. As a user, I want to see a profile page with enrolled courses and recommended courses.
9. As a user, I want to be able search for courses based on provider, source, and subject.
10. As a user, I want to be able to search for courses across platforms by name.
11. As a user, I want to have a user profile where I can see courses I've marked as "taking" (i.e. I have clicked that I am enrolled in) and be linked directly to the course page on the course provider's website.
12. As a user, I want to be able to "hide" (dislike) courses I do not like so that they do not show up in recommendations anymore.
13. As a user, I want to see a short description about the course and some more details like how much it costs, how long it lasts, etc. when I click on a course on CourseOwl as a popup in search results and profile page recommendations.

14. As a user, I want search suggestions to show up when I search CourseOwl for courses.
15. As a user, I want to see recommended courses on my profile page.
16. As a user, I want to be able to close and delete my CourseOwl account.
17. As a user, I want to see courses from iversity, a course provider.
18. As a user, I want to be able to search Google Helpouts for help with a course.
19. As a user, I want to see a clear and understandable categorization of courses into subjects.

Use Cases

Enroll in course:
Primary Actor: Registered user
Goal in Context: The registered user adds a course to profile
Scope: CourseOwl course information page
Level: User
Stakeholders and interests:
Registered user **-** Wants to add a new course to their profile
Recommendation engine - Updates recommendations with new course
Database - Associates course with user
Precondition - User is logged in
Minimal guarantees: Add a valid course as being completed or presently enrolled
Trigger: User selects option to enroll on course page
Main success scenario:
1. User: selects to enroll in course
2. Database: Insert course into user's courses
3. Recommendation engine: Updates similar course for user
Alternative scenario:
1. User: Attempts to enroll in course
2. Database: The user has already completed the goal
3. Database: course not added
4. User: notified of prior enrollment

View Course Details:
Primary Actor: Registered User
Goal in Context: To view details about a course in the CourseOwl database.
Scope: Component
Level: User goal
Stakeholders and Interests:
Registered User - See details about a course, such as the instructor and subject.
Precondition:
User is registered

Minimal guarantees:
User can view correct details about a course on a web page.
Alternative Scenarios:
If a page is not found, show a friendly error message.
Trigger:
1. User searches for a course or browses courses on CourseOwl.
2. User clicks on the name of a course they are interested in on the CourseOwl website.

View Course History:
Primary Actor: Registered User
Goal in Context: To see a list of courses the user has taken or is currently subscribed to along with recommendations
Scope: Component
Level: User goal
Stakeholders and Interests:
Registered User - to see what courses they've taken
Precondition:
User is registered.
User is subscribed to at least one course or has marked at least one course as 'taken'
Minimal Guarantee (success scenario):
User sees a list of courses he's subscribed to or has taken and recommended for.
Alternative Scenario:
If the user isn't subscribed to any course, and hasn't marked any course as 'taken', leave course history blank and recommend random courses
Trigger: User clicks the button 'view profile
1. System displays the user's course history or prompts there is no history.

Sync Courses From Providers:
Primary Actor: Database Manager
Goal in Context: Synchronize the courses from the various course providers with CourseOwl
Scope: CourseOwl backend
Level: Sub-function
Stakeholders and interests:
Registered User - Wants to know when new courses are available and updated
Database Manager - Needs to store new courses
Recommendation Engine - Needs to have full list of courses to recommend
Precondition: Course providers are up and running
Minimal guarantees: New courses that are not already in CourseOwl will be added
Triggers: Time-based trigger, initiated by database manager after specified delay
Main success scenario:
1. Database Manager: Initiates course update task

2. Database Manager: Queries course providers and retrieves course list
3. Database Manager: Adds and updates courses according to providers

Alternative scenarios:

1. Database Manager: Initiates course update task
2. Sync: Queries course providers and retrieves course list
3. Sync: Providers are down, course list not retrieved

Search CourseOwl Courses:

Primary Actor: Unregistered/registered User
Goal in Context: View available courses through website
Scope: Component
Level: User goal
Stakeholders and Interests:
Unregistered User - Search for and view desired courses
Registered User - Search for, view, and sign up for desired courses
Precondition: Courses are synced to database
Minimal Guarantees:
Unregistered user can sign up for CourseOwl after finding desired course
Registered user can add desired course to their CourseOwl profile.
Trigger: User types in query into search box and clicks search
Main success scenario:

1. Search query matches courses in CourseOwl, list of courses displayed to user
2. User can choose courses to view more information or to sign up

Architecture & Design

Overview

We used the Django development framework for the project, backed by a PostgreSQL database instance. As suggested by Django Software Foundation, we divided the project into different apps for the frontend, backend, the JSON API, and the recommendation engine.

Our database includes a common model for all of the courses from different providers, along with a user profile containing information about what courses you have enrolled, completed, and disliked.

In order to import the courses into CourseOwl, we have automated scripts that scrape the providers and insert the course information into the database.

For the front end, we take advantage of Django's templating system to reuse code for headers and footers.

Here is an overview of our Django applications folders ("apps"):

`accounts` - User account and profile management, sign up, log in
`api` - JSON API for use by the frontend website
`courses` - Course scrapers, recommendation engine, database models
`website` - Frontend website, including HTML and JavaScript

UML Diagram

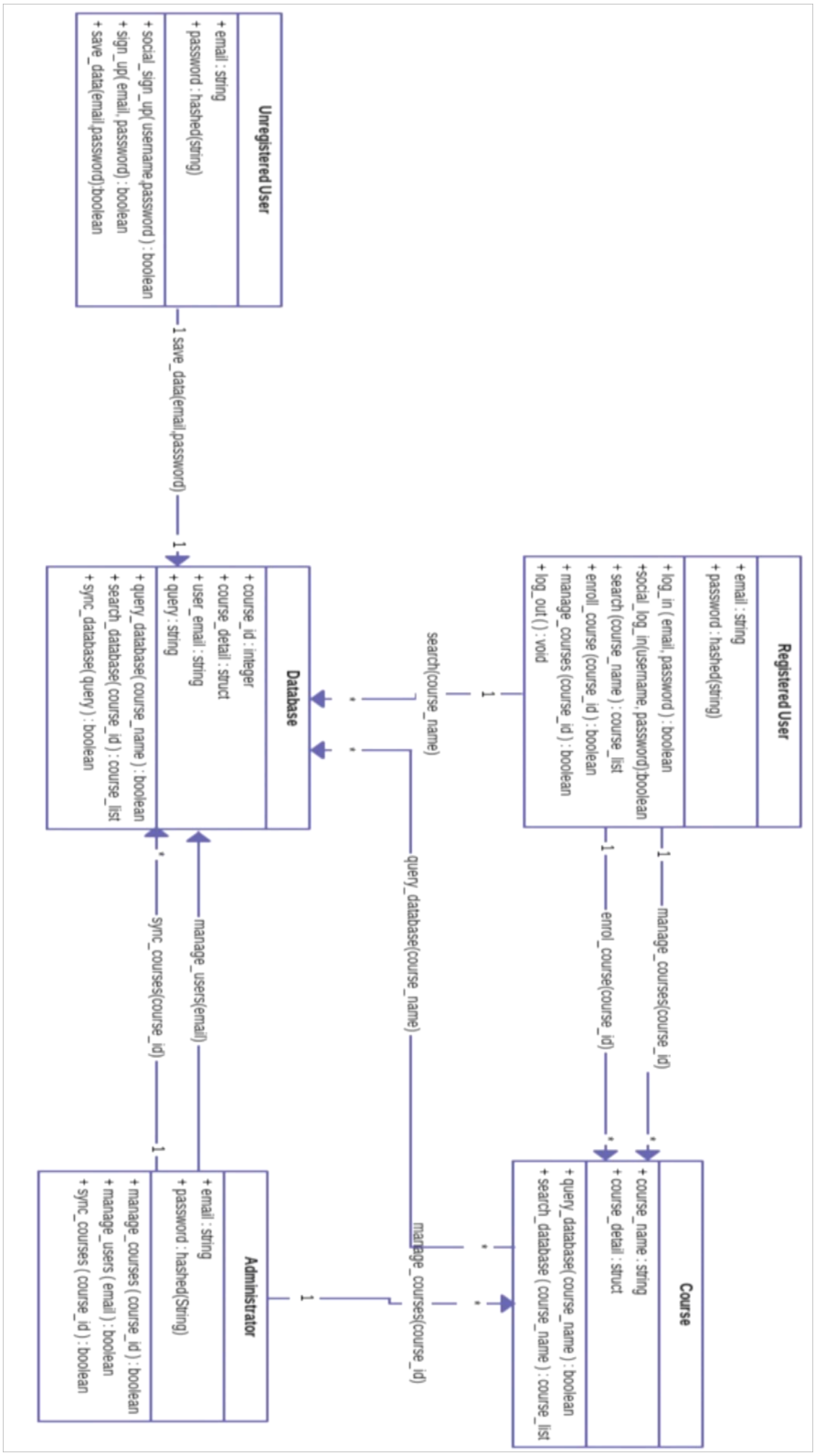

Fig. 1
UML Class Diagram

Future Plans

We have gained a lot of experience by developing CourseOwl. Beyond gaining a deep understanding of the Django web framework, we have learned valuable insights about how to efficiently manage a team of software engineers and delegate responsibilities. Having a structured development process such as Scrum allows team members to develop a sense of personal responsibility and allows us to track contributions and issues over time.

In addition, writing extensive tests allows us to quickly check for errors and run regressions as we are iterating quickly. Our continuous integration system along with issue tracking on GitHub has made testing and deploying CourseOwl fairly quick and painless, and will certainly play a role in our future projects.

We are unsure of the future of CourseOwl given that we will no longer be able to meet together as a team in the future. Regardless, the semester has been a great learning experience in software development. The code is available for the public to view, improve, or learn from, on GitHub.