Student Knowledge System

Team Hive, Fall 2024

Aum Palande, Nishant Basu, Mukil Senthilkumar, Susheel Vadakkekuruppath, Prakhar Suryavansh, Raunak Kharbanda, Ellika Mishra

Summary

Professors have lots of students. A professor should be able to remember what they thought of the student when they took a course, especially when a student asks for a Letter of Recommendation, a TA position, or anything else. The Student Knowledge System is a way for professors to be able to connect with their students more, and allows for the professors to remember legacy students. The stakeholders, professors, should be able to view previous students, as well as view their photo, see comments about what the professor thought about them, and see when students took specific classes. The application meets these requirements, as professors are able to upload rosters directly from Howdy, and then edit comments about them which are saved in a database. The professor should be able to filter and find their students easily. Similarly, the professor can also use quizzes in order to learn the names and faces of the students in their current sections.

To improve the system, we began by fixing minor bugs and updating the upload mechanism to match the new Howdy portal format, ensuring compatibility going forward. The quiz feature was enhanced to allow filtering by specific courses, sections, and semesters, with an updated UI for a better experience. Statistics like total number of correct answers and current streak of correct answers were also added to the home page. Search filters were expanded to include name, email, and UIN, enabling professors to locate individual students more efficiently. The notes feature was updated to a text pop-up format for easier viewing and editing. Together, these updates ensure the system is more user-friendly and effective, helping professors track, organize, and engage with their students both past and present.

User Stories

Sprint 1:

- Deploy New MVP - 2 points:
    - Complete: A new deployment of the MVP was made with working OAuth
- Change login page - 2 points:
    - Complete: The login page was updated to more clearly explain the magic link and input text formats.
- Fix Rspec tests - 1 point:

    - Complete: The old rspec tests that were failing were updated to ensure that they passed.
- Fix Cucumber tests - 1 point:
    - Complete: The old cucumber tests that were failing were updated to ensure that they passed.
- Fix Code climate - 2 points:
    - Complete: The code climate was improved to a B.
- Update docs for setup - 1 point:
    - Complete: The docs were updated for websites that have had updates, and the process was made easier to follow.
- Limit image upload file type - 1 point:
    - Complete: File type inputs were restricted to be png, gif, jpeg or jpg only.
- Create Account Page - 2 points:
    - Complete: The create account page was updated to expect the professor to enter their name, email and department which are used to create the account.
- Students Page to display Students - 3 points:
    - Complete: The students page was fixed to ensure that when new students were created, they would be displayed on it.
- Create Student validation - 2 points:
    - Complete: The create new student page was fixed to require certain fields like name and UIN, and additional checks were added to ensure that the data entered is correct, such as the length of UIN or validity of the email.
- Reduce rubocop offenses - 1 point:
    - Complete: The number of rubocop offenses was reduced to less than 200.

Sprint 2:

- Explain what the export from Howdy more clearly and write upload instructions - 1 point:
    - Complete: The instructions on the upload page were updated to match the data format from the new Howdy page.
- Reduce rubocop offenses - 2 points:
    - Complete: The number of rubocop offenses was reduced to less than 100.
- Change Create New Student Page - 2 points:
    - Complete: Fields like major and classification were updated to dropdowns instead of textboxes to increase clarity and standardize the notations used.
- Change import script for csv file new howdy - 3 points:
    - Complete: The upload controller was updated to match the csv file from the new Howdy data.
- Scrape Images from html file - 3 points:
    - Complete: The upload controller was updated to obtain and store the image data from the .htm file of the new Howdy data.
- Merge the images with corresponding students in csv - 2 points:
    - Complete: The upload controller was updated to ensure that the images from the htm file were matched to the correct students parsed from the csv file.
- Route user to be on homepage in active session - 3 points:

    - Complete: The home controller was updated so that when users were logged in and navigated to the root page they would remain logged in and be redirected to the home page.
- Delete Button for students and courses table - 3 points:
    - Incomplete: A delete button was added that could clear the entire database. However, the implementation was not completed.
- Code coverage - 2 points:
    - Complete: The code coverage was increased to over 90%

Sprint 3:

- Delete Course button - 2 points:
    - Complete: Added button to delete course from database.
- Students Searchable by any field (name, UIN, email) - 2 points:
    - Complete: Modified students page to allow for searching for students by name, email and UIN.
- Add filter for current student/previous student - 2 points:
    - Complete: Added functionality to the quiz to choose between current and previous students to be quizzed on.
- Filter the quiz by courses - 2 points:
    - Complete: Modified the quiz to allow professors to choose specific courses to be quizzed on, instead of all students.
- Code coverage - 2 points:
    - Complete: Code coverage maintained at 90%.
- Make tag not mandatory - 1 point:
    - Complete: Fixed bug where tag was set as mandatory field when creating a new student.
- Fix Dropdowns for Create Student - 1 point:
    - Complete: Fixed bug where dropdowns were not showing when creating a new student.
- Fix default values and refactor/complete upload script - 2 points:
    - Complete: Continued work on upload to fix minor issues relating to default values for empty fields in the csv.
- Fix double commas on the name imports - 2 points:
    - Complete: Fixed bug where two commas were being displayed in the names of students due to an issue in the csv parsing.
- Create note pop-up for professor, give option to edit note - 2 points:
    - Complete: Added option to view and edit notes on the main students page that displays a pop-up text box that can be easily viewed and edited.
- Fix the update student button to post - 2 points:
    - Complete: Fixed bug where changes to update students weren't being saved.
- Fix the apostrophe bug - 3 points:
    - Complete: Fixed bug where students with an apostrophe in their name weren't being parsed correctly from the csv.

Sprint 4:

- Quiz specific sections - 2 points:

    - Complete: Modified the quiz to allow professors to choose specific sections to be quizzed on, instead of all students.
- Fix note issue - 1 point:
    - Complete: Fixed bug related to display of edit notes option on the students page.
- Server error on quiz refresh - 2 points:
    - Complete: Fixed bug where quiz would crash on refresh.
- Statistics for quiz (streak) - 2 points:
    - Complete: Modified quiz implementation to include statistics for streak of correct answers.
- Drop down of current tag, or create new one - 3 points:
    - Complete: Modified tag creation option to allow for choosing between an existing tag and creating a new one.
- Statistics for quiz (total questions) - 3 points:
    - Complete: Modified quiz implementation to include statistics for number of total answers.

Process of Understanding Existing Code

During the first sprint, our primary focus was to understand the existing codebase and address any issues that arose. We began by carefully following the setup instructions in the initial readme, which allowed us to get the application running locally as well as on Heroku. As we encountered challenges with the setup and code, we updated the readme for clarity, adding steps or explanations to address the issues we faced.

Once we had the application running, we focused on fixing immediate issues and resolving bugs that were hindering progress. By the end of the sprint, the entire team had a solid understanding of the codebase, and we were ready to focus on adding new features and improvements in subsequent sprints. The largest updates were made to the upload controller to work with the new Howdy data and to the quiz related code to match the requirements of our client.

Team Roles

1. Sprint 1
    a. Product Owner: Nishant Basu
    b. Scrum Master: Aum Palande
2. Sprint 2
    a. Product Owner: Mukil Senthilkumar
    b. Scrum Master: Susheel Vadakkekuruppath
3. Sprint 3
    a. Product Owner: Prakhar Suryavansh

    b. Scrum Master: Ellika Mishra
4. Sprint 4
    a. Product Owner: Raunak Kharbanda
    b. Scrum Master: Ellika Mishra

Sprint Iterations and Story Points

Sprint Number	Points Completed	Summary
1	18	UI Changes, fixing legacy test cases, deployment, CRUD student operations
2	21	Upload script for new Howdy portal, quiz changes, CRUD course operations
3	24	Quizzing feature, filters to search for students, refactoring and fixing upload script for bugs, note features for students
4	13	UI changes, bug fixes of upload and quizzing features, fix legacy tag features, code coverage

This is in the format of story points (stories completed)

Person	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Total
Aum	3 (2)	6 (3)	5 (3)	4 (2)	18
Nishant	6 (4)	8 (3)	9 (4)	5 (2)	28
Mukil	5 (3)	7 (3)	6 (3)	2 (1)	20
Susheel	5 (4)	7 (3)	4 (2)	2 (1)	18
Prakhar	3 (1)	5 (2)	2 (1)	5 (2)	15
Raunak	2 (1)	5 (2)	4 (2)	3 (2)	14
Ellika	1 (1)	2 (1)	4 (2)	2 (1)	9

Customer Meetings

1. Meeting 1: September 25th at PETR426 from 4:15pm to 5:30pm
    a. This was the initial meeting with the client, and established a time frame and basis for the project. The main portion of this meeting was to meet with the client

and define what we wanted to accomplish per sprint and as an end goal. Our overarching goals of the project were to update the backend for the new Howdy portal, make large UI changes for CRUD operations and cleanliness, and fixing the quizzing features.

2. Meeting 2: October 9th at PETR426 from 4:15pm to 5:30pm
    a. This was the meeting following the first sprint. In this meeting, we demoed the features we made that sprint, including making previous test cases pass, changing lots of UI changes to make login easier, and fixing the students for CRUD operations.
    b. Now that we were more familiar with the codebase and how we manage the project, we decided on focusing lots of resources of Sprint 2 to updating the website to work for the new Howdy portal.
3. Meeting 3: October 23rd, at PETR426 from 4:15pm to 5:30pm
    a. This was the meeting following the second sprint. We demonstrated the new upload script working for Howdy, but it still had some issues with certain classes which we decided we would work on for the next sprint. We also showed our updated UI changes and CRUD operations on courses
    b. We knew we would have to fix and refactor the upload script for the backend in sprint 3, and we also started working on the quizzing features.
4. Meeting 4: September 6th, at PETR426 from 4:15pm to 5:30pm
    a. This was the meeting after the third sprint. In this meeting we showed the fixed upload script which was working for all different data sets. We did extensive testing with the client and found no other issues that were easily seeable. We also added a notes and tag feature, where the professor can edit and view notes on their students. Finally, we also showed some changes on the quizzing portion of the code.
    b. To make the code future proof and better for the next group, we wanted to put good time into code coverage and fixing test cases.
5. Meeting 5: September 20th, at PETR426 from 4:15pm to 5:30pm
    a. In this final meeting after sprint four, we showed lots of UI changes on the quizzing feature. Similarly, we also made lots of backend and bug changes, as well as all our passing cases and code coverage.

BDD/TDD Process

Behavior-Driven Development (BDD) is a valuable approach for ensuring that the features being implemented meet the client's needs. In our project, we successfully created user stories that captured key client requirements, such as spaced repetition and passwordless login, and broke those stories into actionable tasks for the team. However, a limitation we encountered with BDD was managing the numerous small requests from the client. These requests often had to be deprioritized in favor of larger, more time-intensive features.

Test-Driven Development (TDD) also proved beneficial. For instance, during the development of the course filtering feature, our Cucumber tests effectively validated the user stories. They were

clear enough to identify and fix bugs without needing to run the application. Additionally, during the migration refactor, these tests ensured that the refactor did not break existing functionality. However, we did encounter some challenges. Some tests, like those for spaced repetition, were written after the feature was implemented, reducing the effectiveness of TDD. Despite this, the Cucumber tests provide long-term value by allowing future teams to refactor and add features confidently, knowing that existing functionality remains intact.

Rspec, on the other hand, presented more challenges for our team. Some Rspec tests, particularly those for helper functions in course filtering, felt redundant as they overlapped with Cucumber tests, which seemed unnecessary given Agile's emphasis on delivering working code over rigid processes. The late switch to passwordless login also made updating Rspec tests challenging. Furthermore, the legacy codebase used Rails tests instead of Rspec, requiring us to rewrite tests entirely to meet the professor's requirement for high Rspec coverage, which somewhat undermined the goals of TDD.

Despite the challenges, we appreciated the opportunity to develop and refine our testing strategies. The lessons we've learned from BDD, TDD, and Rspec will be invaluable as we move forward into industry projects.

Configuration Management

The main way we managed our code was through Git and Github. Our project was hosted publicly, and we were all collaborators on the same project. We all worked in our separate branches, and made sure to pull into main every so often to not have large amounts of merge conflicts. Some of our work was overlapping, so to ensure that the code was working before we pushed to main, those two people would combine their branches together first and handle conflicts. We tried to keep the number of commits and pull requests to a minimum, with the main branch protected. In order to merge into main, the pull request would need to be approved by one other person not working in the same feature. We also tried to pull and merge all our branches a day or so before the deadline, which meant that we could resolve conflicts with ample time. We had to do a spike in the last sprint, when code that was pushed had overridden some other features. This was because the base of the branch was a much earlier version of main, and thus it had not been in the branch changes. We had a couple people trying to figure this out at the same time, and eventually we were able to fix it by rebasing the branch onto the top of main, handling conflicts in the side branch, and creating the pull request to make sure all the conflicts were handled. In the end, we had 62 branches and 78 pull requests, because at the beginning of the project, we planned on deleting stale branches, but eventually figured it would be ok to leave them alone.

Heroku Deployment

During the production release process to Heroku, we encountered several challenges that required troubleshooting and fundamental changes to address effectively. The following outlines the issues faced, how they arose, and what steps we took to resolve them.

1. Git Configuration on WSL
Our development environment utilized Windows Subsystem for Linux (WSL), and setting up Git credentials within this environment proved challenging. While following the official guide for Git integration with WSL (https://learn.microsoft.com/en-us/windows/wsl/tutorials/wsl-git), we encountered issues with the Credential Manager. For some unknown reason, the manager failed to function as intended within the WSL subsystem. This prevented us from seamlessly managing credentials for Git operations.

Resolution:
To overcome this, we generated a Personal Access Token (PAT) and configured Git to use it manually. This bypassed the dependency on the Credential Manager and allowed us to authenticate and push our code to the repository without further issues.

2. Session Management in the Quiz Module
One of the key features of our project was a new quiz module that initially relied on session-based storage to manage user data. During deployment, we discovered a fundamental incompatibility with Heroku's architecture. Heroku dynos, which are ephemeral by design, do not share sessions across dynos. As a result, our session-based solution started throwing errors, making it unusable in the Heroku environment.

Impact:
This issue rendered our initial architecture for the quiz module non-functional and required us to re-evaluate our approach entirely.

Resolution:
To address this, we redesigned the module to move away from session-based storage. Instead, we implemented a more scalable solution using a centralized database or other persistent storage mechanism to manage user data. While this redesign required additional effort, it ultimately ensured compatibility with Heroku's architecture and enhanced the robustness of our solution.

Tools and Gems

1. GitHub: GitHub is a popular platform for version control and collaboration, built on Git. It allows teams to manage code repositories, track changes, collaborate through pull requests, and streamline development with integrations like CI/CD pipelines.
2. AWS Deployment: AWS (Amazon Web Services) provides cloud infrastructure for deploying web applications. Services like EC2, S3, and RDS enable scalable, secure, and flexible deployments. AWS is widely used for hosting and deploying both static and dynamic web applications.
3. Google Auth Deployment: Google Authentication (Auth) integration allows users to sign in using their Google accounts. It's commonly used to implement secure, passwordless authentication in web applications, improving user experience and security.

4. SimpleCov: SimpleCov is a code coverage analysis tool for Ruby projects. It helps developers track which parts of their code are tested by generating detailed reports on the coverage of each file, making it easy to identify untested areas.
5. Code Climate: Code Climate is a platform that provides automated code quality analysis and maintainability insights. It integrates with version control systems to track code changes, offering feedback on complexity, duplication, and potential issues.
6. Tailwind CSS (Gem): Tailwind CSS is a utility-first CSS framework that allows developers to style web applications quickly. The Tailwind gem integrates it with Ruby projects, enabling rapid prototyping with minimal custom CSS.
7. OmniAuth (Gem):OmniAuth is a flexible authentication system for Ruby applications that allows developers to implement third-party login via providers like Google, GitHub, and Facebook, simplifying the integration of OAuth-based logins.
8. Nokogiri (Gem): Nokogiri is a powerful HTML, XML, SAX, and Reader parser for Ruby. It is commonly used for web scraping and parsing XML/HTML documents, providing a simple interface to extract and manipulate data.
9. Passwordless (Gem): Passwordless is a Ruby gem that enables password-free login systems. It allows users to authenticate using email or other verifiable methods, providing a secure and user-friendly alternative to traditional password-based logins.

Necessary Links

Pivotal Tracker: https://www.pivotaltracker.com/n/projects/2721032
Github Repository: https://github.com/nishant-basu-tamu3/student_knowledge_system
Application: https://hive-sks-c9da748e2491.herokuapp.com/
Code Climate Report:
https://codeclimate.com/github/nishant-basu-tamu3/student_knowledge_system
Presentation Video: https://www.youtube.com/watch?v=QuoQf-Wqbe8
Demo Video: https://www.youtube.com/watch?v=QuoQf-Wqbe8

Repository Contents

- Project Structure :
  /home/nishantbasu/csce606-project/student_knowledge_system/

```
├── .github/
│   └── workflows/
│       └── build.yml
├── .gitignore
├── Gemfile
├── Gemfile.lock
├── README.md
├── app/
│   ├── assets/
│   │   ├── builds/ (.keep)
│   │   ├── config/ (manifest.js)
│   │   ├── images/ (TAM-Logo.png, k2.png, etc.)
│   │   └── stylesheets/ (application.scss, application.tailwind.css)
│   ├── controllers/
│   │   ├── application_controller.rb
│   │   ├── courses_controller.rb
│   │   ├── students_controller.rb
│   │   └── users_controller.rb
│   ├── models/
│   │   ├── application_record.rb
│   │   ├── course.rb
│   │   └── student.rb
│   └── views/
│       ├── courses/ (_form.html.erb, show.html.erb, etc.)
│       ├── students/ (quiz.html.erb, index.html.erb, etc.)
│       └── layouts/ (application.html.erb)
├── config/
│   ├── application.rb
│   ├── environment.rb
│   ├── routes.rb
│   ├── puma.rb
│   └── initializers/ (assets.rb, omniauth.rb, etc.)
├── db/
│   ├── migrate/
│   │   ├── 20230228194732_create_courses.rb
│   │   ├── 20230228194940_create_students.rb
│   │   └── 20241108013359_create_quiz_sessions.rb
├── lib/ (course_entries.rb, student_entries.rb, etc.)
```

└── public/ (.keep)

- The Github ReadME has the exact steps mentioned to get the application running locally and the steps needed to deploy the application to Heroku.
- The steps to set up the Google Oauth and the AWS bucket are also mentioned in the readme file.