Final Report: Student Knowledge System

1. **Two paragraph summary of the project as implemented, including the main customer need and how the application meets it, including who the stakeholders are. This will contrast to what you wrote in Iteration 0.**

Class sizes can have over a 100 students and that is just in one semester. Over the course of numerous semesters it gets harder and harder for professors to keep track of their students. The goal of this application, also known as the Student Knowledge System, is to allow professors, the stakeholders, to learn the names of their students in relation to their faces so that they are better equipped to identify and engage with them in a classroom setting. Many times students who had a professor a few years ago ask for a recommendation letter, but professors have a hard time remembering how that student was in their class. In the application, data on every student and every course is saved for that user, so they can go back at any time and look at student notes if they ever have to write a recommendation letter.

Our application meets the main need of the customer, as they wanted an improved method of practicing students' names rather than through a quiz. One main feature that is implemented is the idea of spaced repetition. Professors will get a set of students to be practiced that will dynamically change if the professor matches the students' names to their faces correctly. This spaced and repetitive practice is a strategy that leads to superior long-term learning and memorization. They also wanted to be able to more securely login to the application and not deal with passwords. Thus our website features a passwordless authentication that authenticates through the user's email. Though there are a few user stories that we were not able to implement, we prioritized which ones mattered most to the customer. Overall, the customer wanted a more streamlined and well designed website to use regularly to learn about their students and all of the features we implemented and the refactoring we did contributed to the improvement of the usability of this application.

2. **Description of all user stories (including revised/refactored stories in the case of legacy projects). For each story, explain how many points you gave it, explain the implementation status, including those that did not get implemented. Discuss changes to each story as they went. Show lo-fi UI mockups/storyboards you created and then the corresponding screen shots, as needed to explain stories.**

User Stories: Login
As a user,
So that I can access the application in a more secure manner,
I want to be able to authenticate without a password.

Task Description	Implementation Status	Points
Send a magic link to the console for local authentication	Complete: The magic link was implemented by updating the user model, mounting an engine in the routes file, writing a method in application_record.rb, updating the mail credentials, and	2
Update production environment to send	Complete: The mailer needs credentials	3

email to the user so that they can access the application	from the shockwavekickers606@gmail.com in order to send emails. These credentials were added.	
Style login and related pages to the clients expectation	Complete: Removed toolbar and all other buttons besides sign in and create account on all login related pages	3
Remove login via devise entirely from application (cleanup files)	Complete: Devise is an authentication gem that was replaced with the passwordless gem.	1
Cucumber test cases	Complete: Cucumber test cases were implemented to test the login. Special attention was given to the “When I login” step that required creating a session manually.	1
RSpec cases for user authentication	Incomplete: Due to the challenges of having to retroactively write Rspec cases for all of the Legacy Code (since the group we inherited from did not use Rspec), we were not able to complete these rspec cases.	1
Implement OAuth login	Incomplete: After successfully completing passwordless login via MagicLink, fully implementing a second login system was not necessary to make a viable final product.	3

User Stories: Courses
As a user,
So that I can find a specific course within my course history,
I want to be able to have a simple way to filter my courses.

Task Description	**Implementation Status**	**Points**
Filter courses so that professor can more easily access the one they are looking for	Complete: Filtering courses according to a course name, the semester it was offered, and what students were in the course allows for a comprehensive search system	3
Modify the courses page so that course	Complete: The course history button was	1

history is more accessible	somewhat hidden in the legacy code which made it unclear how CRUD operations on a course would work. Thus the button was moved and styled.	
Reformat student view within courses for better filtering and viewing	Complete: The table that was present in the courses tab was restyled.	1
Create cucumber cases for courses	Complete: Cucumber tests that verify the functionality of the course page and the various searches has been implemented	1
Create rspec cases for courses	Complete: Writing specs that verify the functionality of CRUDing courses, as well as verifying the functionality of the model and controller functions that help with the search feature were implemented.	2
Filter according to multiple searches at once	Incomplete: Due to other features taking priority later in the project, this feature was placed in the icebox.	1
Style courses related pages	Complete: Using tailwind, the course pages were styled which includes the index, profile page, history pages, and CRUD operation pages.	3

User Stories: Students
As a user,
So that I can view my students in a customized way,
I want to be able to see my students in different views, and have a more organized summary of each student.

Task Description	**Implementation Status**	**Points**
Fix the mismatching of students' names to their faces when a user uploads a .zip file from howdy	Complete: The legacy code had an implementation that matched the order of the .csv student information to the content in the order of students in the images folder. These did not match and names and faces were mismatched. Parsing through the .html page was done instead to get the correct order of images.	2
Create a yearbook style view so professors can just see images and	Complete: In order to see the student's faces better and have a view more aligned	1

names rather than all student information	with the applications main focus of learning student's names and faces, the student index table was reformatted.	
Create a toggle between yearbook style and student list view so that user can decide if they want to use computer energy to load larger images	Complete: The client did not like the yearbook style view so a toggle button was made to show either a tabular or yearbook style view.	2
Improve UI for all student related pages to be more accessible and functional	Complete: CRUD operation pages specifically had a UI that was difficult to use so these were reformatted.	2
Create cucumber cases for students	Complete: Cucumber test cases were made to make sure the students have functional CRUD operations.	1
Create rspec cases for students	Complete: Writing specs that verify the functionality of CRUDing students was completed	2
Change upload page to include better instructions and cleaner UI	Complete: The upload page was fixed to have instructions pop up when the user clicked a button.	1
Deleting multiple students at once	Incomplete: Due to the volume of students that this app is likely to accumulate over time, this is a good feature to have, though it is not high enough priority to our client to complete in our limited time frame	1
Create Student Data for quiz and demo use	Complete: we modified the seed.rb file to generate hundreds of students in a handful of courses, so as to give us sample data for demoing and testing features	1
Style student related pages	Complete: Using tailwind, the student pages were styled which includes the index, profile page, and CRUD operation pages. Each page was given one point.	3

User Stories: Quizzes/Dashboard
As a user,
So that I can more effectively and efficiently learn students' names,

I want to use a spaced repetition method to match students' names to their faces and have a simple way to access the feature.

Task Description	**Implementation Status**	**Points**
Add columns to Database	Complete: This required modifying the schema of the database via migrations to include a column that tracks the last time a student was practiced, and a column to track how much time until the student should be practiced again.	**1**
Access to Spaced Repetition Quizzes	Complete: This required adding a "practice now" button on the dashboard that appears when students are due to be practiced and disappears otherwise, and redirects to a student's quiz page	**1**
Remove quiz functionality and tab from application, and all corresponding parts	Complete: Done in two phases, first by removing quiz-related features from the front end, and second, by deleting all code that relates to quizzes and modifying the schema of the database to not have Quizzes or Qrosters	**2**
Add timer for each student for next time that student needs to be practiced	Complete: under student info, the time until a student is due to be practiced is displayed.	**1**
Update time data when student quizzed	Complete: this required implementing a simple algorithm that doubles or halves a student's interval based on whether they were guessed correctly, and changes the last practiced at value to the current time	**1**
Create a Student Quiz Page	Complete: This required modifying the original format of a quiz page to be functional as a page associated with a student, rather than a quiz, and modifying the rails routes and UI accordingly	**2**
Quiz continues until all students guessed	Complete: this required creating and managing an array that keeps tracks of due students, randomly selecting due students to display next, and automatically redirecting to the dashboard when complete	**1**

Quiz bug: redirect to login page	Complete: after passwordless authentication was complete, the quiz was redirected to the login page instead of the dashboard. This bug was fixed.	**1**
Quiz bug: duplicate answer choices	Complete: due to a mistake in how quiz pages are generated, sometimes a student's name would appear as two answer choices. This bug was fixed	**1**
Create cucumber cases for spaced repetition	Complete: Cucumber test case was made to make sure spaced repetition worked as expected.	**1**
Create rspec cases for spaced repetition	Incomplete: Because of the challenge of writing specs for the legacy code, these specs were not developed by the end of our project.	**1**
Quiz bug: Occasional duplicate quiz pages	Incomplete: This baffling bug, where the quiz sometimes selects the same student to display twice in a row, is a flaw in the spaced repetition that we did not have the time or knowledge to solve. I theorize it has to do with the updating of some values after an answer is made updating too slowly.	**2**

User Stories: Refactoring

These user stories are not specifically what the professors will see but rather what will help the next group when they take over this legacy project.

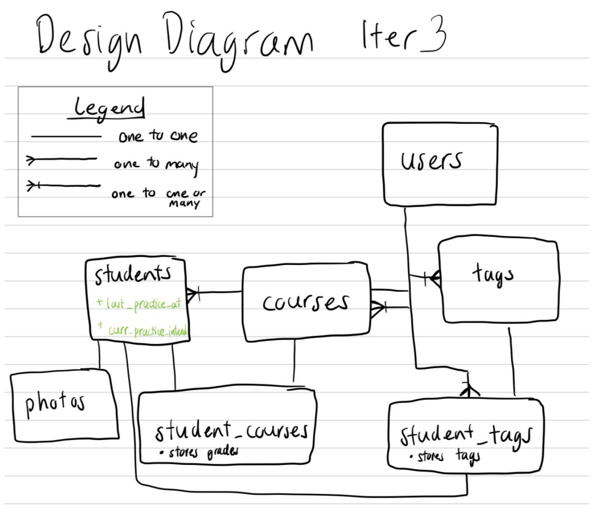
Task Description	**Implementation Status**	**Points**
Consolidate disorganized migrations so that it is more clear as to how the schema is generated	Complete: The legacy code had migrations that became a problem after the pg -> sqlite3. The migrations were consolidated to match the current schema.rb at the time.	**1**
Modify the database.yml file in order to use sqlite3 instead of postgres for the test and development environment	Complete: The gem pg was used for all environments. This was advised against as it would slow our process. A refactor was made so that the lighter, quicker sqlite3 was used in test and development. This process broke a few things that had to be handled.	**1**

Reorganize gemfile to include gems in their respective environments	Complete: Gems were originally not organized according to environment and we organized them by test, development, and production. Mainly, testing gems like cucumber, capybara, were moved to testing only. Sqlite3 for development and testing, and pg (postgres) for production.	**1**
Fix foreign key errors with database	Complete: Changing from pg to sqlite3 caused errors with foreign keys. This error halted our team and was fixed.	**1**

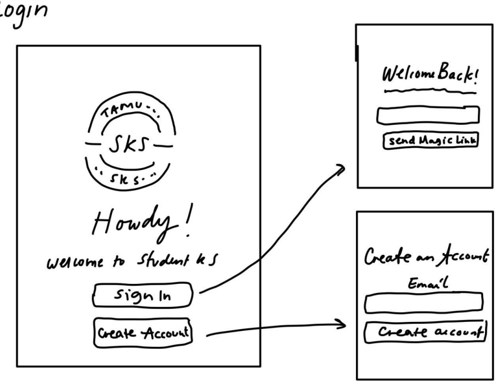
Final Deliverables

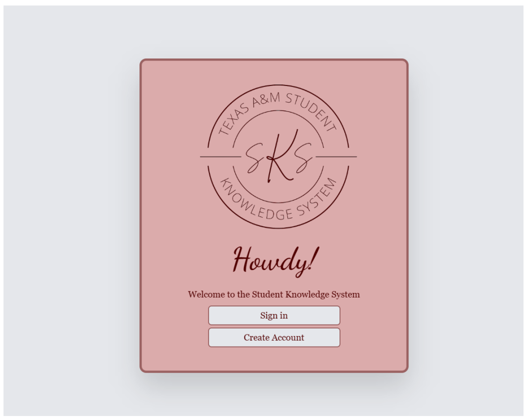
Task Description	**Implementation Status**	**Points**
Final Document	Complete	**1**
Final Presentation Poster	Complete	**2**
Customer Satisfaction Survey	Complete: The survey will be sent to the client to be filled out after the final presentation.	**2**
Final Deliverables Setup	Complete: Organizing the final deliverables was no small task.	**2**
Final Presentation Video	Complete	**2**
Final Document: Writing Sections	Complete	**2**

Mockups:



Login





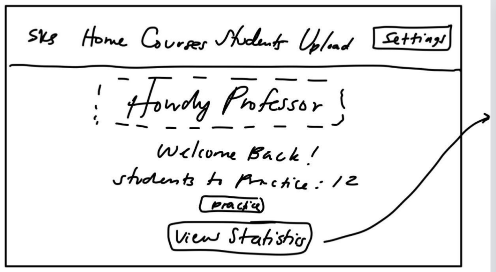
Howdy!

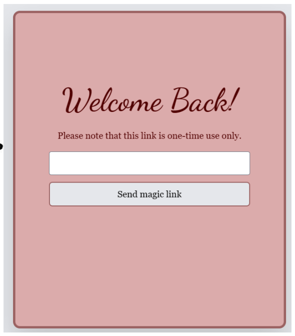
Welcome to the Student Knowledge System

Sign in

Create Account

Dashboard

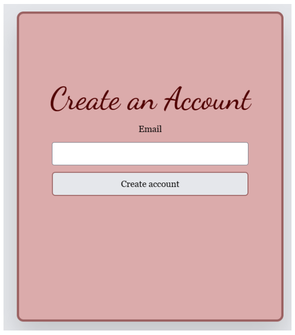




Welcome Back!

Please note that this link is one-time use only.

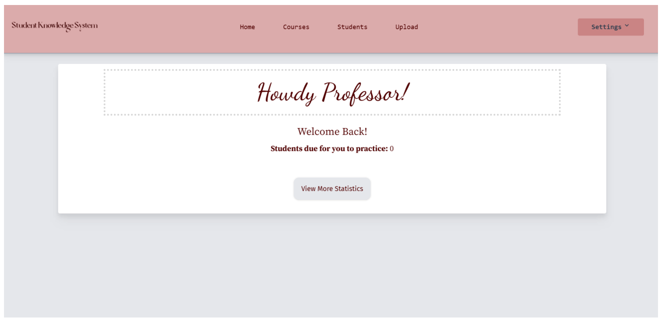
Send magic link



Create an Account

Email

Create account



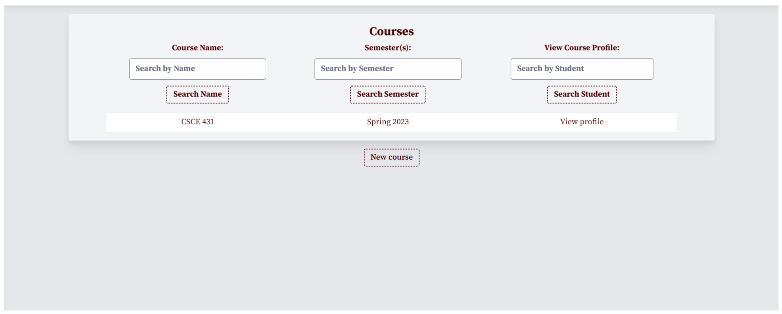
Student Knowledge System Home Courses Students Upload Settings

Howdy Professor!

Welcome Back!

Students due for you to practice: 0

View More Statistics



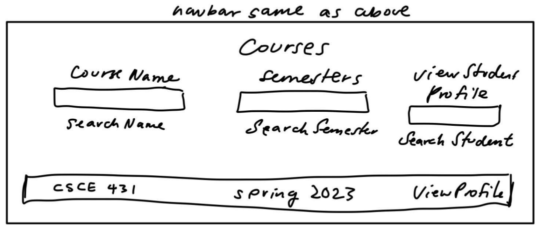
Courses

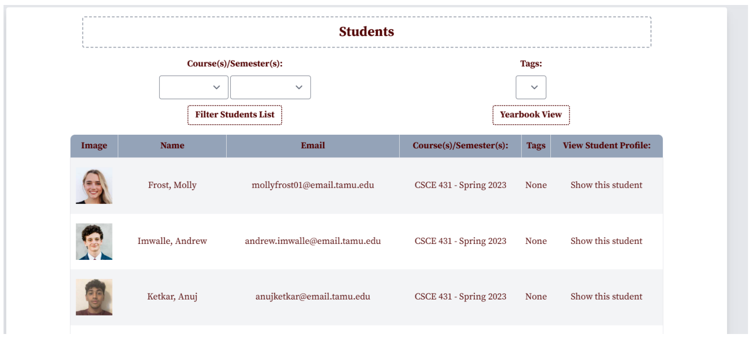
Course Name:	Semester(s):	View Course Profile:
Search by Name	Search by Semester	Search by Student
Search Name	Search Semester	Search Student
CSCE 431	Spring 2023	View profile

New course

Courses

navbar same as above





Students

Course(s)/Semester(s):

Tags:

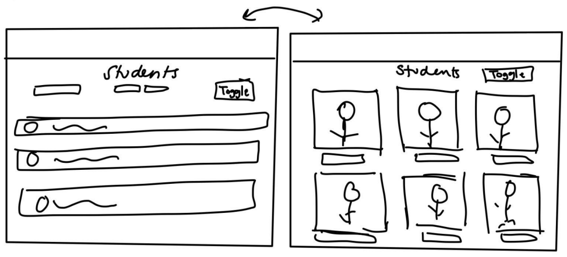
Filter Students List

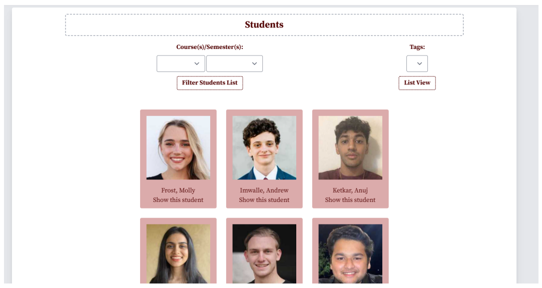
Yearbook View

Image	Name	Email	Course(s)/Semester(s):	Tags	View Student Profile:
	Frost, Molly	mollyfrost01@email.tamu.edu	CSCE 431 - Spring 2023	None	Show this student
	Imwalle, Andrew	andrew.imwalle@email.tamu.edu	CSCE 431 - Spring 2023	None	Show this student
	Ketkar, Anuj	anujketkar@email.tamu.edu	CSCE 431 - Spring 2023	None	Show this student

Students

switch back & forth





Students

Course(s)/Semester(s):

Tags:

Filter Students List

List View

Frost, Molly
Show this student

Imwalle, Andrew
Show this student

Ketkar, Anuj
Show this student

3. **For legacy projects, include a discussion of the process for understanding the existing code, and what refactoring/modification was performed on the code, in addition to the user stories listed above.**

The process of understanding the existing code began with deploying the legacy code. In doing so, we understood how to get the code functioning and were able to step through the current application, view all the pages, and play with features that were working. In doing so, we found many features that were functioning as well as a few bugs. The CRUD operations for each model were working as expected. A bug that was found was the name and face mismatch. Reading through the code base and examining the file tree was also done. We found that the models did not utilize the rails active record associations. Thus, a database refactor was explored but decided against since the team did not possess sufficient knowledge of ruby on rails to refactor a large part of the code base in a time-efficient manner. Rather, we delved into writing test cases and implementing new features. Through this process of testing and implementing, we grew to understand the existing code. User stories that we worked on such as organizing gems, switching database gems, implementing spaced repetition, and writing test cases were especially helpful in understanding the code. Furthermore, some features were not easily deployed from the existing code such as mailing. Fixing this feature to work helped in understanding how sending mail works in ruby on rails.

Throughout our team's time with the code, the main changes from the existing code occurred with the view styling, the upload controller, rewriting test cases from rails to rspec and cucumber, migrations refactoring, testing and development database gem changed to sqlite3, and removing quizzes. Each element in the views was styled with tailwind. The upload controller had a bug fix and the database was configured so that we could store student photos. The rails tests and migrations from the previous team were scrapped and replaced. Lastly the existing gemfile was organized in addition to including a few more gems.

4. **List who held each team role, e.g. Scrum Master, Product Owner. Describe any changes in roles during the project.**

There were brief changes in roles from iteration to iteration, depending on who was best equipped to take the lead for the biggest action items in a given iteration (i.e. spaced repetition for i3, ui changes for i5, etc.) We have listed the roles from iteration to iteration below:

Iteration 1: *Harshitha Dhulipala* : Product Owner + Developer, *Neha Manuel* : Scrum Master + Developer, *Ethan Novicio* & *Caleb Oliphant*: Developer

Iteration 2 *Harshitha Dhulipala* : Product Owner + Developer, *Caleb Oliphant* & *Neha Manuel*: Developer, *Ethan Novicio* : Scrum Master + Developer

Iteration 3: *Caleb Oliphant* : Product Owner + Developer, *Neha Manuel* & *Harshitha Dhulipala*: Developer, *Ethan Novicio* : Scrum Master + Developer

Iteration 4: *Caleb Oliphant* & *Harshitha Dhulipala*: Developer, *Neha Manuel* : Scrum Master + Developer *Ethan Novicio* : Product Owner + Developer

Iteration 5: *Harshitha Dhulipala* : Product Owner + Developer, *Caleb Oliphant* & *Neha Manuel*: Developer, *Ethan Novicio* : Scrum Master + Developer

5. **For each scrum iteration, summarize what was accomplished and points completed.**

Iteration	**Summary of Accomplishments**	**Points**
0	Iteration 0 was dedicated to understanding the legacy code base and running all the functionality of the project. The main goal before writing the user stories which we did at the end of this iteration, was to run the legacy code both locally as well as deploying it to heroku. We also created our plan for the UI changes and listed some general user stories and features we wanted to add after talking to the customer. Some of the main user stories we noted down based on customers' needs was simplifying the dashboard, filtering courses, spaced repetition, passwordless/google login, and modifying the UI to make it more accessible.	**0**
1	Iteration 1 was dedicated to understanding the database and creating a design diagram for how we want to design and refactor some of the code. We even spent a lot of the time familiarizing ourselves with rspec and cucumber tests and learning more about BDD/TDD. In this iteration we were able to have the deployed app running and were able to run the test cases using their testing framework rails test.	**3**
2	Iteration 2 saw our first new features, as well as some crucial bug fixes and support that would assist in the testing of the new features. First, one of the most inhibiting bugs we noticed from the legacy project, the mismatched faces and names of students during a file upload was fixed. The three different kinds of course filtering were implemented, and seed data was generated to display this functionality during a demo. Cucumber tests were written to verify the functionality of the courses as well. Additionally, there was a lot of bad design when it came to how the migrations were laid out, so we did a lot of planning in regards to how we want our schema file and database to be formatted.	**7**
3	Iteration 3 was our first major change to the Student Knowledge System. With our client emphasizing the importance of implementing spaced repetition as he envisioned it, we overhauled the current quiz system and implemented spaced repetition with its respective rspec and cucumber test cases. Additionally, we had been having some issues with the display of uploaded pictures, and those changes were fully fixed this iteration. We also implemented yearbook style here so that the client can view the students in different views.	**12**

4	With this iteration, we made several refactors to assist with our development for the remainder of the project. We transferred our development environment from Postgres to SQLite, reorganized the gemfile, refactored the migrations and fixed database bugs, meanwhile, the early steps toward implementing passwordless login began to take shape as we experimented with the passwordless gem and magic link. Finally, significant progress was made on adding Rspec tests to the legacy code.	**9**
5	With the time dwindling to complete our project, we made a final push to make major improvements to the application by finishing the implementation of our passwordless authentication using a one-time email link and overhauling the UI using tailwind. In addition to these major changes, we fixed bugs in spaced repetition and Rspec tests that arose from the new login system, and improved cucumber test coverage to cover the majority of our app.	**32**

6. **List of customer meeting dates, and description of what happened at the meetings, e.g. what software/stories did you demo.**

Customer meeting notes and details per iteration are shown below, as well as a description of what we discussed

Iteration 1:
Time: 11:00am - 12:00am, Date: 1/6/2023, Place: Ibis Bencoolen Singapore

Notes needed: Harshitha was product owner

Iteration 2:
Time: 12:45pm - 1:15pm, Date: 1/9/2023, Place: Ibis Bencoolen Singapore
Notes:

- Move the welcome backup as a popup and make it one line so that the dashboard has a more clean and simplistic design for the professor to access
- Fix mismatching of names when the professor uploads zip file from howdy
- Fix decimal places on quiz statistics to be 2-3 decimals for score
- Add a streak (longest streak of getting students correct) on dashboard
- Add accuracy statistic on each of the student within student page and the highest one on the dashboard
- View as pictures and view as rows so that user has the option to not load large images
- Write a python script to generate data in order to test and view web page functionality

Iteration 3:
Time: 4:15pm - 4:45pm, Date: 1/18/2023, Place: Online via Zoom
Notes:

- Create an Admin user that can CRUD users
- Edit student Notes outside of Edit Student notes (bigger text box too)
- Make yearbook style look good when students don't have pictures

- Course view has too much empty space due to scroll
- Image upload in Edit Student is broken
- Spaced Repetition in quizzes
    - Home page: __ many students due to review, click to review
    - Course info: options for review due and all students in given course
- Client MUST be able to log in as Dr. Ritchey

Iteration 4:
Time: 3:00pm - 3:30pm, Date: 2/22/2023, Place: In Person Peterson
Notes:
In this client meeting, the website with the temporary login feature was presented. After successfully signing up and logging in, the client moved through the application and suggested UI changes. One of these changes was to improve the search options to show the sections available in a drop down. Later, the client wanted the student index page to be displayed in a table format.The spaced repetition feature was also presented. The client liked the implementation and suggested clearer information on screen such as removing the practice button if there are no more students to practice. A few spaced repetition bugs were also discovered and have been added as user stories for iteration 5.

Iteration 5:
Time: 3:00pm - 3:30pm, Date: 2/22/2023, Place: In Person Peterson
Notes:
In this client meeting, the website with the temporary login feature was presented. After successfully signing up and logging in, the client moved through the application and suggested UI changes. One of these changes was to improve the search options to show the sections available in a drop down. Later, the client wanted the student index page to be displayed in a table format.The spaced repetition feature was also presented. The client liked the implementation and suggested clearer information on screen such as removing the practice button if there are no more students to practice.

7. **Explain your BDD/TDD process, and any benefits/problems from it.**

Behavior Driven Design is an excellent way to ensure that the features the group is working hard to implement do satisfy a need of the client. We were able to come up with user stories that accurately portrayed some of the needs of our client, such as spaced repetition and passwordless login, and translate those stories into points that could be split up among the team. The only downfall of BDD in our project was that our client often made a large number of small requests, which often were pushed into low priority due to the larger features we had to spend greater time perfecting.

Regarding Test Driven Development, we saw benefits from this practice. During the development of course filtering, for example, the cucumber tests accurately tested the user stories in an interpretable enough way that we were able to fix the bugs course filtering had without ever running the app. Furthermore, during the migration refactor, the implemented cucumber tests allowed us to test and verify that our refactor was working and not breaking the application. However, some of our Cucumber tests, like our tests for spaced repetition, were written after Spaced Repetition was implemented, which took away from some of the effectiveness of writing the tests to begin with. Looking to the future, cucumber allows teams building upon the code base to safely refactor and implement new features. Running the cucumber tests will verify that any new features or refactors are not affecting existing features.

Rspec, unlike Cucumber, was largely problematic for our team. Firstly, with Rspecs for the helper functions in course filtering, some Rspecs seemed to be testing the same functionality as the Cucumber test, just in a different way, which felt like an extra and not super helpful step when considering Agile's prioritization of producing working code over following a set-in-stone process. Additionally, because we ended up switching to passwordless so late in the project, modifying the test cases beforehand to account for the passwordless gem was quite challenging. Additionally, because our legacy code used rails test instead of Rspec for its testing, we basically had to scrap their tests and write our own test for the legacy code, meaning that our professor's request to have high rspec coverage for our app defeated the purpose of TDD. Even with all of the ups and downs we were satisfied with the opportunity to develop tests for code, and look forward to taking the lessons we learned here to industry.

8. **Discuss your configuration management approach. Did you need to do any spikes? How many branches and releases did you have?**

We used Git for our configuration management, cloning the repository locally and making changes in branches so as to avoid compromising the functionality of the master branch. Once features were complete, branches were merged to master. There were several spikes over the course of our project, usually toward the end of iterations as we tried to push our app to a minimum viable state for the software. The most notable spikes occurred during iteration 3, when the spaced repetition was implemented, and iteration 5, when the UI was modified and the magic link began working successfully. Over the course of the project, we performed 5 releases, one at the end of each iteration, and our work was spread out over 15 branches throughout the project.

9. **Discuss any issues you had in the production release process to Heroku.**

The process of releasing to heroku was fairly straightforward. Following the directions and support that Heroku provided for ruby on rails development was sufficient. Although, a few issues did occur and their solutions will be shared here. Heroku login was not working so "heroku login -i" was used instead. Providing your account api key can replace providing a password. It is possible to set up an ssh key to login.

After entering "git push heroku master", if the terminal stalls then make sure the ssh key is configured. If "heroku keys" shows no keys then "ssh-keygen" will create a key that you can put in the .ssh file. Then, "heroku keys:add" adds the generated key. Furthermore, make sure that "git remote -v" shows that you are pushing to the correct remote. Our project showed this result.

```
heroku  https://git.heroku.com/student-knowledge-system.git (fetch)
heroku  https://git.heroku.com/student-knowledge-system.git (push)
origin  git@github.com:nehamanuel/team_cluck.git (fetch)
origin  git@github.com:nehamanuel/team_cluck.git (push)
```

Lastly, these can change with `git remote set-url heroku https://git.heroku.com/student-knowledge-system.git`

10. **Describe any issues you had using AWS Cloud9 and GitHub and other tools.**

Github was used for version control and collaboration, and ran smoothly for the most part except for some occasional hiccups caused by poor commit and merging practices, as well as some confusion on how the software works. Firstly, there were a couple of times when two people completed the same user story because one had not merged the branch that they had been working in, resulting in the second person thinking the story still needed to be implemented.

Additionally, sometimes features that needed to be implemented to work on other features were not merged to master, which slowed down progress as the creator of the unmerged branch would need to be contacted so they could merge the branch. Finally, there was a binary file called "courses" that always caused merge conflicts, and, though not a huge problem, made GitHub more cumbersome to use.

One of the issues that one of our members, Caleb, ran into was, as the project was reaching a close at the end of iteration 5, his Virtual Machine he was using for development ran out of storage and, as a result, he was unable to run his code locally for his development and testing. The solution to this problem is to buy more storage, but because the project was so near completion, we did not consider this to be worthwhile, and Caleb's remaining stories were picked up by others.

11. **Describe the other tools/Gems you used, such as CodeClimate, or SimpleCov, and their benefits.**

Passwordless: a gem used to authenticate a user by sending them a link. It was beneficial in removing passwords from the authentication process and creating sessions. Sessions keep track of the user's login and access to the website. It was not beneficial in testing since the passwordless_sign_in() function did not work as expected.

Tailwind: a gem for implementing fast css styling.

Omniauth: a gem that provides access to login with google.The path to "/auth/google_oauth2" leads to sign in.

SimpleCov: a gem that shows how much of your code is covered by test cases.

CodeClimate: a gem that shows the quality of your code.

Nokogiri: a gem for parsing through html.

12. **Make sure all code (including Cucumber and RSpec!) is pushed to your public GitHub repo.**
13. **Make a separate section discussing your repo contents and the process, scripts, etc., you use to deploy your code. Make very sure that everything you need to deploy your code is in the repo. We have had problems with legacy projects missing libraries. We will verify that everything is in the repo.**
    Read the github readme
    setup.sh is a script to prepare your machine.
14. **Links to your Pivotal Tracker, public GitHub repo, and Heroku deployment, as appropriate. Make sure these are up-to-date.**

    Github Repo Link: https://github.com/nehamanuel/team_cluck

    Pivotal Tracker Link: https://www.pivotaltracker.com/n/projects/2595510

    Heroku Link: https://student-knowledge-system.herokuapp.com/

    a. **Make sure the instructor (pcr@tamu.edu) is an owner of the project on Pivotal Tracker. ✔**
15. **Links to your presentation video and demo video.**
    https://drive.google.com/file/d/1PKgJ2TJITWbY-cSCBnb22LYxxP790Gkv/view?usp=sharing

Try to limit the report to no more than 6 pages of 12-point text, not counting figures (e.g. screen shots and GUI). If you need to more space to document everything, that is okay. It must be in PDF format.
WRITE THE REPORT AS IF IT IS BEING READ BY A TEAM WHO WILL FOLLOW ON TO YOUR PROJECT. INCLUDE WHAT YOU WOULD WANT TO KNOW IF TAKING OVER THE PROJECT. IF YOU NEED MORE SPACE, USE IT.
Note that you are responsible for ensuring that all project material necessary facilitate future work on your project is uploaded to the GitHub repository for your project or is otherwise available to both the instructor and the next team(s).