CSCE 606: Lightfoot

Team members: Colton Simpson, Jacob Kelly, Kunal Vudathu, Rahul Shah, Stuart Nelson

Final Documentation

- **Team roles**: The team should elect a scrum master and product owner.
    - We have an agreed upon rotation that will shift with each iteration
    - Scrum Master: Jacob Kelly
    - Product Manager: Rahul Shah
- **Customer meeting date/time/place**: The team product owner should contact the customer and set up a regular weekly meeting.
    - Every Friday at 2pm in PETR
- **SCRUM meeting date/time/place**:
    - Every Tuesday and Thursday at 10am
- Github: https://github.com/Lightfoot-Heavy-Machinery/shockwave_kickers
- Pivotal Tracker: https://www.pivotaltracker.com/n/projects/2595510
- Heroku Deployment: https://shockwave-kickers.herokuapp.com
- Slack Workspace name:
    - Workspace: tamu.slack.com
    - Channel Name: csce606_shockwave_kickers

I. Project Summary

Our client requested a Student Management System to allow them to remember and keep track of students for an extended period of time. They wanted the application to maintain a record of all previous courses they have taught, including sections, and the corresponding students. The client also requested a quiz minigame to help associate a student with their appearance.

To that end, we have built Shockwave Kickers, a Student Management System that allows a user, primarily a Texas A&M Faculty member, to store their courses and students in the application. Both courses taught and the students within those courses are kept track of and have basic information stored about each. The application also contains the quiz functionality as described by the client, as well as an upload feature to allow quick transfer of student information from Howdy to the application.

II. User stories Manifest

- **Ability to view course profile** (To view relevant information for courses)
- **Student Quiz** (To help the professor remember the names of students)

- **Ability to view all student records** (To provide an overview of students in a given course)
- **Ability to view student profiles** (To view what courses students have taken)
- **Ability to enter/store class rosters** (To view and edit a list of current and previous students)
- **Landing Page** (To provide an smoother user experience)
- **Deploy app to Heroku** (To allow the user to access the app remotely)
- **Create base app** (To provide structure to the features we provide in the app)
- **Student records CRUD**
- **Course records CRUD**
- **Parse CSV file**
- **Create database schema** (To design a database with tables and columns, identifying primary keys)
- **React template** (This was not used in the final version)
- **Semester records CRUD**
- **Base UI template** (Design a UI that is consistent with the wireframe)
- **Stitch frontend and backend**
- **CI/CD Pipeline (Not accomplished)**
    - Integrate CI/CD pipeline into Github and Heroku
- **Update db schema on backend and perform changes/migrations as needed**
- **Backend: Implement db update on json received by parse csv function**
    - We are now able to parse the CSV roster file that is loaded from
    - Automatically builds Course and Student records, and write these records to the database
    - If a student is in a course that doesn't exist in the database, then the system will automatically create a new Course record before saving the Student
- **Implement Authentication**
    - Users can sign up, get email confirmations, reset passwords emails.
    - We also implemented the need to confirm email before signing in.
    - All the pages will be accessible only when the user has signed in.
    - A user can access only the information that they created, everything is private.
- **Migrate React to html and build pages to what figma diagram looks like**
    - Build a base template for all the pages
    - Courses page, student profile page, course profile page were built and stitched with backend and authentication
    - Edit student and edit course were created to what it looks like figma
- **Implement S3 bucket where photo is stored in AWS and integrate with all controllers**

    - Each student model needs a picture attached, so we used S3 bucket to store these images as blobs where each image is attached to each student object in postgreSql
    - As part of the upload process, each student will have an attached picture uploaded.
    - The work involves creating S3 bucket and making all controllers/views compatible with the S3
    - Implemented image transformation as all images should be of the same size
- **Create Quiz Backend/Frontend functionality**
    - This story implemented quiz view and CRUD operations
    - This work involves implementing the basic quiz algorithm to show a student image and show multiple names of students in the class and the user has to select one of them.
    - The work also tracks progress made on the quiz
    - You can create a quiz and get back to the incomplete quiz. View history of quizzes.
- **Dynamically the populate dropdown menus in the Course page**
    - When users want to filter the list of students in a certain class, the user will select filters from 3 dropdown menus based on semester, section and tags. These dropdown menus are now populated dynamically based on values in the database, without duplicates.
- **Dynamically the populate dropdown menus in the Student page**
    - When users want to filter the list of students in a certain class, the user will select filters from 3 dropdown menus based on semester, section and tags. These dropdown menus are now populated dynamically based on values in the database, without duplicates.
- **Backend: Dynamically generate file names to order photos to students based on csv**
    - First step in the upload algorithm is to parse the csv and attach photos to the respective customers. This is what this story is about, it makes sure correct photos are linked to the respective students.
- **Upload with real sample data**
    - Get an actual sample CSV to test our parsing
- **(Misc fixes) Fix duplicate success message popup on quiz creation**
    - When a quiz was created/saved, the success/fail message would appear twice
    - To improve the UI/UX, this needed to be changed
- **Filter student list on the students page**

    - A user should be able to filter the list of students by selecting semester, course, and tag filters through the dropdown menus
    - Should work with consolidated student entries (tags, semesters, course, etc.)
- **Link course page students list with the dropdown menus**
    - A user should be able to filter the list of students by selecting semester, section, and tag filters through the dropdown menus
- **Avoid duplicate entries in courses page and students including pictures page**
    - Due to db design, the UI was showing duplicate entries for courses and students, the story was concatenate course entries with same into one entry
- **Implement algorithm and keep track of quiz scores**
    - When a quiz is taken, the user should be able to see their score for the quiz
    - A user should be able to view past quiz scores
    - A user should be able to see how many they've gotten correct in a row on a quiz, and what their longest streak in that quiz has been
    - Users should be able to see how many times they incorrectly guessed a student's name before it was guessed correctly
- **Build home page**
    - A dashboard to provide an overview of this user
    - It will show counts of semesters, classes, and years taught
    - It should show average quiz and streak scores for the most recent semester and overall
    - It should display the most relevant students regarding memorization
    - It should display some of the most recent courses
- **Update edits/destroys of students/courses**
    - The story includes add edit and destroy pages for Courses
        - When deleting a course, all students under the course will be deleted
        - When editing a course, course info across all students in the course will update
    - Added a page for viewing course history
    - For student pages, the story includes:
        - Updating student's grade of that class
        - Updating info across all students
    - Added destroy course of a student functionality where you can delete course attached to a student
- **Implement upload frontend and finish backend work for controller**

    - The upload page UI is finished, and shows the upload button, as well as the upload history.
    - Users can delete previous CSV uploads
    - Backend CSV parsing is done
- **Build CI/CD pipeline using github actions**
    - Built CI pipeline to make sure all unit cases are run before merging as well as committing the change to prod
    - Built CD pipeline to push change to heroku
- **Segregate student table into common table and student course table**
    - Segregate student table into common table for student common data and student course table for student course data
- **quiz and qroster deletes should be done; notes concatenate**
    - As part of student and course deletes, delete respective info of Qroster and quizzes
- **Checks for creating course or course of a student or creating student**
    - Add regex checks on the UI to make sure the fields match expected format. This includes: UIN, email, major, course name, section and so on
- **Implement sort (asc, des) by student last name in course profile**
    - The user can now sort the student list alphabetically or reverse alphabetically for increased ease of use
- **Student profile end to end**
    - We can now view the course history for any student from their student profile, as well as the semester and year taken.
- **Adding quiz scores to the course profiles**
    - The user can now look within course profiles to see how well they are remembering names so that they can further practice courses that have the lowest scores
- **Completing the student upload functionality**
    - When the user provides the application with a zip file containing a csv of student attributes followed by images of each of the students, students will be posted to the database and will appear on the students page and the courses page of the application.
- **Filtering students by search in both the courses and students page**
    - The user is given an increased ability to look for a student profile to check for specific attributes that may be useful for quizzes or for future letters of recommendation.
- **Filtering students by tags**
    - To make it easier to find students with specific notes or attributes, the filter-by-tag feature allows a user to flag a student with custom attributes to

add ease-of-use to name memorization or future letters of recommendation.

- **Implementing user profiles by allowing for password resets**
    - Names for accounts and password reset functionality was added in case a user forgets their password or if certain classes need to be in the field of view of one user and not another. (Increased scalability)

IV. Scrum master and PO every iteration

Iteration	SCRUM Lead	Project Owner
0	Jacob Kelly	Rahul Shah
1	Stuart Nelson	Colton Simpson
2	Rahul Shah	Kunal Vudathu
3	Colton Simpson	Jacob Kelly
4	Kunal Vudathu	Stuart Nelson
5	Jacob Kelly	Rahul Shah

Iteration 0: 16 Points Completed

Iteration 0 was mainly about laying down the foundation for our application. It consisted of designing a Figma chart to represent what a finalized frontend might look like for our application. We also designed a database to work with the backend and made a schema to help visualize the database structure. Finally, we set up a weekly meeting with the client to help communicate our progress and their idea for the final product.

Iteration 1: 11 Points Completed

Iteration 1 was focused on finalizing the foundation laid out for the application in Iteration 0. The software database schema was finalized and the beginnings of a frontend and backend were implemented. Additionally, we also tested these changes on a live server by deploying the master branch of our application to Heroku.

Iteration 2: 0 Points Completed

In Iteration 2, we got the base application up and running, and completed the basic CRUD operations for the Student, Course, and Semester models. Basic CSV parsing and upload was implemented, and we deployed our application to Heroku. We also made final adjustments to the database schema.

Iteration 3: 10 Points Completed

In Iteration 3, we implemented the CRUD operations for the quizzes, as well as some frontend quiz views. We integrated Amazon s3 into the application for storing student images. Semester and Course lookups were implemented in the student table, as well as dynamically populated dropdown filters on the course page.

Iteration 4: 16 Points Completed

In Iteration 4, we finished work on viewing records of both students and courses when we completed the filtering functionality for them. We finished off the work on the Quiz by implementing the backend algorithms that drive the student randomization functionality. We also started work on the final features of our application by building the frontend for both the Upload and Home Page pages.

Iteration 5: 18 Points Completed

In Iteration 5, we finally implemented our CI/CD pipeline. We did final bug fixes and implemented “safe delete” to ensure that deleted students are also removed from quiz rosters, and deleted courses also deletes the students in those courses. We updated the database schema by adding a table containing attributes common across multiple student instances. We upgraded students tags, and implemented the final CSV upload page.

Customer Meeting Dates

We met the customer every Friday from 2pm-2:30pm in PETR 426, beginning on September 23rd, 2022. Our final meeting was on December 2nd, 2022. The only exceptions were October 14th and November 25th.

BDD

We did a Behavior Driven Development process with regards to our software implementation. This entailed us making many user stories for atomic tasks within our application. These stories were then doled out in the first of the bi-weekly team meetings, and then progress was shared on the next bi-weekly meeting. This helped everyone keep track of the rest of the team’s progress. Tests were written for the new code to ensure correctness and to help debugging during future integrations. The completed stories were finally demonstrated in front of the client the week that the iteration was due.

Configuration Management Approach

We used git and GitHub for version control, auditing, accountability, and collaboration. We had the following configurations:

- HTML/Bootstrap Build configuration - Frontend
- Ruby on rails Build/ dependencies configuration - Backend
- Runtime configuration
    - Heroku Dyno runtime
    - Postgres on Heroku configuration
- External configuration
    - AWS Configuration for S3 Bucket (Serving static product image files)
    - Email Confirmation set up via app code

We had 5 major releases, planned and deployed after each iteration.

Issues faced with Heroku:

Since Heroku deprecated the free tier, we had to apply for the student account and get free Heroku credits. This is a temporary fix, as eventually the credits will run out. We had to upgrade postgre db to mini and web server to echo dyno package

Issues faced with AWS:

When implementing S3 bucket for image storage, due to lack of knowledge in AWS S3, we didn't know how to make S3 buckets public and make sure S3 writes were only given to users with IAM permissions.

Gems and additional tools used

- Mini_magick
- Bootstrap
- Postgres
- Aws-sdk-s3
- SimpleCov
- image_processing
- Rubyzip
- dropzonejs-rails

User Documentation
Docs Link (view only):
https://docs.google.com/document/d/1ATG78_72BFUqlMq_9StImvl8vVKKumL87lb0Caz3JoQ/edit?usp=sharing

This document should include a startup guide for both heroku and local setup (windows and mac user). This also contains other instructions to store s3 and send confirmation emails when running locally

Future work

Following is the future work that could be done:

- Implement levels on quizzes
    - First level being 4 options (already implemented)
    - Second level being a text input and they are
    - Third level being a text input and checking whether the name is correct (close to same)
    - Fourth level being same as level three but exact match
- Implement SSO, so you can use a common login
- Today we support student creation but we don't support student addition to a course. So support that by adding a page of adding a student to existing course
- Implement secondary users like TAs where they use the account but can't edit anything
- Implement dynamic search instead of static search. Two changes needed:
    - Currently the user needs to click a button to search when sorting by semester, section in course profile page, or sorting by tag, course name or semester in students page. The future work is to remove button or sort the moment the user clicks on the option
    - Text box search implementation doesn't auto populate the entries as well. We need to click the search button and they have to be an exact match as well.
- In the edit student page, we can add tags to students today. Improve tags selecting UI, where today we need to ctrl click and select tags and separate text box to create a tag. Instead use a much more friendly UI. Very minimal backend changes.
- Multi Selecting students to add tags to all of them instead of going to each student to add tags.
- Adding system tests and integrating them in the CI pipeline.
- Adding smoke tests to CI pipeline and adding a staging cluster to make sure the app doesn't fail in some time.