A Gamified Course Visualization, Organization, and Assessment System

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Thien Alexander Nguyen
Spring 2020.

Technical Project Team Members

Andrew Abraham

Connor Anderson

Jack Herd

Ryan Kann

Corey Lando

Taylor Nelson

Jimmy Patterson

Table of Contents

List of Figures

Abstract

Throughout the technology industry, there are many groups clustered around the pursuit of a good idea, developing an application or new way of solving a problem through rapid technological development. Under the direction of Professor Mark Floryan, this group developed an application designed to augment and re-frame the pedagogical experience of those in introductory and advanced coding classes. By designing the course's interface to coalign the learning experience and the grade-based incentive structure, we sought to qualify the hypothesis that students at a college level can learn more easily when they can complete their coursework at their own pace, and without the burden of fast-approaching deadlines or single-chance quizzes and tests. Using the Agile development methodology, this group took on an existing codebase and modified it to meet the needs of the customer, outfitted with customization for the professor, a working student interface, and elements of automated grading, such as in-course quizzes. After two semesters of development experience, the group learned much about the Agile system, where it fails, and where it succeeds, including how to handle the issue of complete worker turnover, the quirk of frequent customer input, and the difficulty of learning new languages and forms. Using this application, a new type of course will be tested and implemented at the University of Virginia, and hopefully in educational centers across the planet, in an effort to improve the educational experience for both students and teachers.

1. Introduction

Over the course of American History, few technological achievements have been quite as inspiring as the Apollo Program. Through 11 tough years, NASA rocketed to an interstellar stage while facing impossible hardships, deadlines, and pressures. While it would be ridiculous to equate the technical portion of our capstone project to the Apollo Program, it's easy to compare the similarly bumpy roads, quick deadlines, and steep learning curves. While our software will not put a man onto the moon, it will certainly transform learning for all who use it.

1.1 Problem Statement

Our client is Mark Floryan, professor at UVA. He currently develops the curriculum for, and teaches, the Data Structure and Algorithms course in the computer science pilot program. The course's topics are organized in a directed, acyclic graph (DAG) representing the dependencies between them (for instance, *Linked Lists* may depend on *Pointers*, forming an edge from the latter to the former). Each of these topics is assessed individually, and students acquire a status of *incomplete* (not yet learned the material), *competent* (baseline understanding of the material), and *mastery* (strong understanding of the material). Floryan hopes to use a novel algorithm from a research student to judge the level of competency of students based on their assessment results.

Previously, Floryan did not have a clear, dynamic, and practical way to communicate the DAG nature of the course topics to his students. The development of this was the goal of the previous team of the project, and was mostly completed. However, the grading structure was not in alignment with his model, and many security vulnerabilities existed in the system, severely hampering its usefulness as a tool.

Assessment is currently done using traditional paper quizzes. Since students can potentially reassess on any topic in any given week, Floryan is currently giving each student a copy of the quiz for each topic. The students complete the quizzes for the topics they wish to assess, and then all of the pages of each student's quizzes are scanned and graded. This solution has many issues. Firstly, large amounts of paper are wasted. Further, more clerical effort must be expended in the copying, distributing, and scanning of the quizzes than should be required.

Grading of these quizzes is currently done through Gradescope. While a useful tool for traditional assessments, it is not particularly aligned to Floryan's needs. Gradescope assumes a uniform set of assessments for each student of the course, while Floryan's model generally has a distinct set of assessments for each student. The current work-around is to define assignments for each pool of quizzes each week, so each student has the same set of "assignments" in the system; however, this means that the grade on each of these assignments is not directly interpretable as the grade for any topic, requiring further effort to extract useful grading information.

The system we develop will further refine Floryan's abilities to display the DAG structure of the course topics. Additionally, it will greatly improve the assessment system. Students will be able to take their assessments online through the system, meaning that excess unused quizzes will not be generated or submitted. The novel assessment metric will also be integrated directly into the system, removing another layer of grade interpretation. Finally, the system is designed with Floryan's model in mind, so conflicts in assumptions, like those which exist with Gradescope, will be greatly diminished.

1.2 Contributions

At the beginning of the year, we worked to understand and repurpose the best efforts of a predecessor group, which had, over the course of the previous year, developed a working website

with severely limited functionality. Over the course of the year, we were able to polish up the code which we had been given, eliminating extraneous elements, improving APIs, and fulfilling formerly faltering functionalities. In specific, we eliminated a tendency for the frontend code to pull the entire database for each query, which alleviated strain not only on the backend, but also on the frontend. This allows for more independent users to access the site at once, without overwhelming any servers, which in turn makes this product viable for use in large classes. Additionally, we shored up the authentication process, ensuring that all API endpoints were properly secured. Furthermore, we added features to the application, granting more ability to professors to designate how their courses are structured, allowing them to upload students and grades in spreadsheet format (making this application compliant with UVACollab's teacher-facing grade features), and lock or unlock certain topics to students who hadn't been demonstrating expertise. Most importantly of all, we developed a quizzing feature which automates and grades students' assignments in-site, which complies with Professor Floryan's style of giving quizzes as well as efficiently allowing students to see their graded work as it affects their course grades in real time.

2. Related Work

The Student Performance Tracker is based on a few project-based-learning-centered web-based systems. Some examples of programs that fall in this category are McGraw-Hill's ALEKS and Pearson's MyLab & Mastering.

Firstly, McGraw-Hill's Assessment and Learning in Knowledge Spaces (ALEKS) system is "a Web-based, artificially intelligent assessment and learning system. ALEKS uses adaptive questioning to quickly and accurately determine exactly what a student knows and does not know in a course. ALEKS then instructs the student on the topics she is most ready to learn. As a student works through a course, ALEKS periodically reassesses the student to ensure that topics learned are also retained." (McGraw-Hill, 2020) ALEKS was "developed from research at New York University and the University of California, Irvine, by a team of software engineers, mathematicians, and cognitive scientists." (McGraw-Hill, 2020)

One of Professor Floryan's requirements for a web-based system is for the system to utilize a multiple-choice bank of questions which he can pull from. Because ALEKS avoids the use of multiple-choice questions, this system would not satisfy Professor Floryan's needs. Furthermore, students cannot learn Computer Science related subjects using ALEKS, as it only offers mathematics, business, and science courses. (McGraw-Hill, 2020)

Pearson's MyLab & Mastering is another assessment tool based on the idea that students learn at their own pace.The system personalizes study, helping "students understand what they know, what they do not, and where to spend their time studying." MyLab & Mastering has "personalized learning pinpoints the precise areas where each student needs practice, giving all students the support they need, when and where they need it, to be successful." (Pearson Inc., 2020) Pearson's MyLab & Mastering does offer computer science courses, unlike ALEKS.

We have expanded upon the previous team's project. We added the ability for the system to administer quizzes, the ability for professors to designate teaching assistants, proper authentication and security measures, and scalability for professors to create classes of any size.

Quizzes can have multiple-choice or Parson's problem questions, pulling random questions from a bank created or uploaded by the professor. Each question has a predetermined answer key, and the system can automatically grade quizzes. Quizzes can also administer free-response or coding questions if the professor wants, but they cannot be automatically graded and must be manually graded by the professor or their teaching assistants.

Professors can designate teaching assistants, allowing them to access and modify student grades. Teaching assistants can also grade other students' submissions manually if needed for free-response or coding questions.

The system allows students to log in, properly authenticates them using Google's OAuth protocol. In addition, the system's API endpoints are now properly secured and cannot be accessed without being authenticated first. The system allows for mass upload of both rosters of students and their grades, via comma-separated-value files.

3. System Design

At the highest level, this application is designed to allow professors to set up classes, administer homework, and give quizzes to their students, all while tracking grades and seeing overall progress. Students as well have the ability to take their quizzes, see their grades, and interface with the class through this system.

We developed the frontend of this system with Vue.js (Javascript) and the backend through Django (Python). These are the frameworks and languages in which our foundational code was written, passed up to us from a group which worked to build a less robust version of this system last year. We are currently operating under the GNU GPL version 3. This license secures the rights of anyone to use and change our code, and to share that code and any changes made to it. It also contains a variety of defenses against malicious legal action involving the license itself.

3.1 System Requirements

Gathering system requirements allows designers to work directly with the customer to create metrics that clearly display what end goals the customer has for the product. Requirements also allow the designers to visualize progress during the development of the product. Assigning each requirement a deadline forces the designers to plan the order that each requirement will be fulfilled, streamlining task scheduling. Creating a comprehensible list of deliverables with the customer lets the designers know what functions and features they will need to plan to implement.

Our minimum requirements:

Professors should be able to:

- Access all the functionality of the system.

- Prevent student data from being accessed by other students or unidentified users.

- Set up topics to follow the syllabus.

Our desired requirements:

Professors should be able to:

- Administer multiple choice questions.

- Administer Parson's Problem questions.

- Have the system pull questions randomly from a bank.

- Display the final grade with stair-step level of competency per topic.

- Have the system perform well and not show signs of noticeable delay when submitting assignments and viewing results.

- Upload data and have it be processed quickly.

- Give access to TAs so that they can access student grades

Students should be able to:

- Access their own data and grades for the course.

- Only display units that have been 'unlocked'.

Our optional requirements:

Professors should be able to:

- Set up automatic grading for each topic to match the grading system of the syllabus.

- Administer short-answer/coding questions.

- Allow graders (Profs/TAs) to manually assign grades to longform answers.

3.2 Wireframes

At a base level, our application is very UI heavy. There is a large quantity of information to constantly create, read, update, delete, and display to the user. With that, wireframes are essential, as they ensure that both the client and developers have a single clear and concise outline of how the data in the application should be presented to the user. Luckily, a large portion of our data is structured in very similar fashions, making the wireframing process substantially modular.
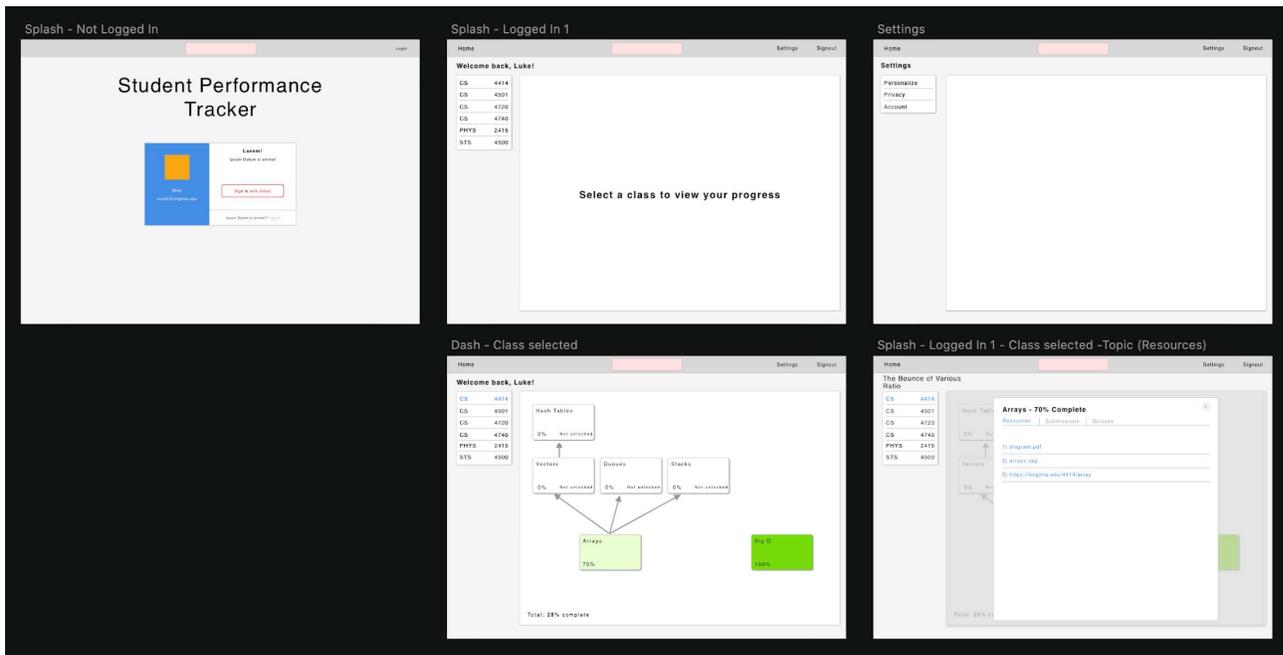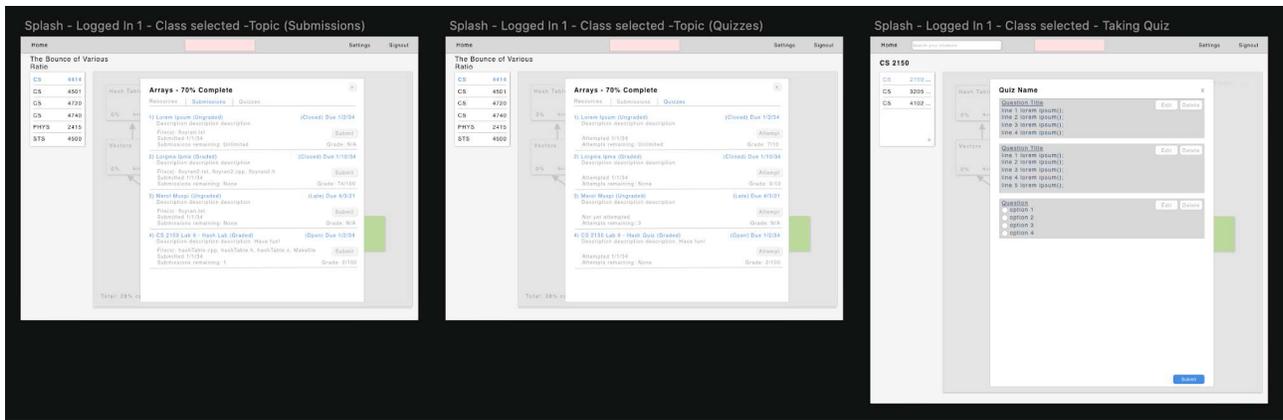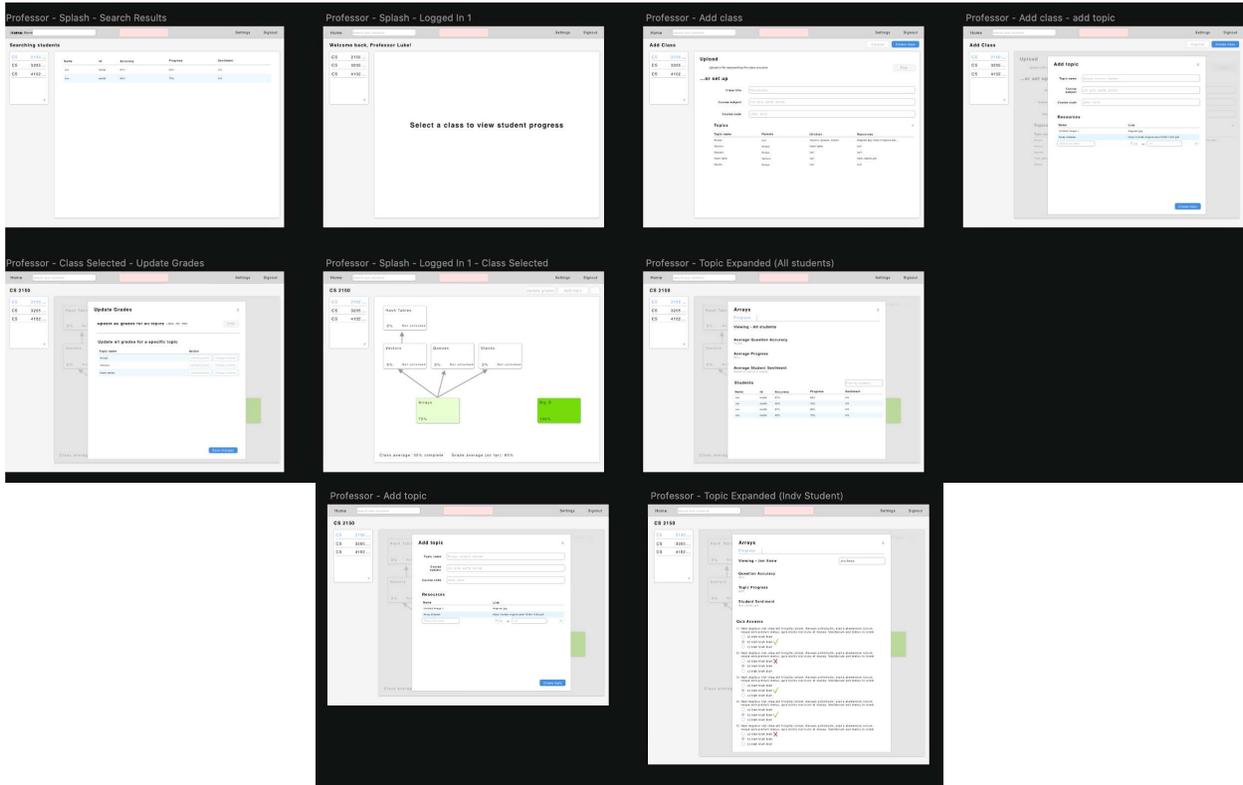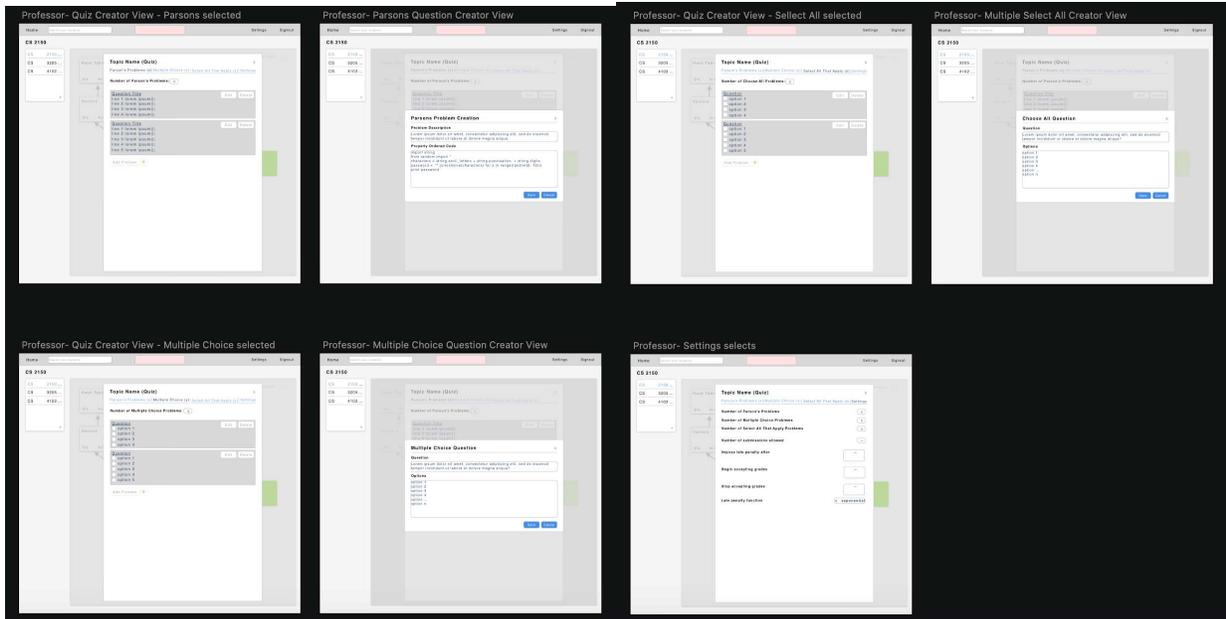
Figure 1: Wireframes 1

Figure 2: Wireframes 2

Figure 3: Wireframes 3

3.3 Sample Code

Figure 4: Models Sample Code

These three code blocks are the Quiz, Student to Quiz, and Quiz Question models. The quiz

feature was one of the ones most extensively worked on by this team.

```
'''

    Quiz
────────────────────────────────────────
'''

class QuizViewSet(viewsets.ModelViewSet):
    authentication_classes = [GoogleOAuth]
    permission_classes = [IsAuthenticated & ( IsProfessor | ReadOnly)]
    renderer_classes = (JSONRenderer, )
    queryset = Quiz.objects.all()
    serializer_class = QuizSerializer
    model = Quiz

    '''
    | url: DELETE :: <WEBSITE>/api/quiz/<quiz-question-id>
    | function: Removes a given question
    '''

    def delete(self, request, format=None, pk=None):
        if pk is None:
            return missing_id_response()
        else:
            try:
                result = self.model.objects.get(pk=pk)
                result.delete()
            except self.model.DoesNotExist:
                return object_not_found_response()

            return successful_delete_response()
```

```
'''
| url: GET :: <WEBSITE>/api/quiz-question/ OR
|     GET :: <WEBSITE>/api/quiz-question/<quiz-question_id> OR
| function: Retrieves all or a single quiz questions
'''

def get(self, request, format=None, pk=None):
    is_many = True

    if pk is None:
        quiz_id = request.GET.get('quizId', None)
        if quiz_id is not None:
            try:
                quiz = Quiz.objects.get(pk=quiz_id)
                # result = self.model.objects.get(pk=quiz.pk)
                result = self.model.objects.filter(assignment=quiz.assignment)
            except Quiz.DoesNotExist:
                return object_not_found_response()
        else:
            #TODO What should the default response contain without topic id?
            result = self.model.objects.all()
    else:
        try:
            result = self.model.objects.get(pk=pk)
            is_many = False
        except self.model.DoesNotExist:
            return object_not_found_response()

    serializer = self.serializer_class(result, many=is_many)
    return successful_create_response(serializer.data)
```

```
'''
| url: POST :: <WEBSITE>/api/quiz-question/
| function: Creates a given quiz-question
'''

def post(self, request, format=None, pk=None):
    if pk is None:
        data = json.loads(request.body)
        # If topics are not at the root but instead inside the topic key, use that instead
        if 'quizzes' in data:
            data = data['quizzes']

        # Topic becomes Assignment
        # Course becomes Topic
        for quiz in data:
            print("")
            print("Quiz:")
            print(quiz)
            print("")
            serializer = self.serializer_class(data=quiz)
            if not serializer.is_valid():
                return invalid_serializer_response(serializer.errors)
            serializer.save()

            assignment = Assignment.objects.get(pk=quiz["assignment"])
            newly_created_quiz = Quiz.objects.get(
                assignment=assignment,
                pool=quiz["pool"],
                practice_mode=quiz["practice_mode"],
                allow_submissions=quiz["allow_submissions"],
                next_open_date=quiz["next_open_date"],
                next_close_date=quiz["next_close_date"],
            )
            students_in_assignment = StudentToAssignment.objects.filter(
                assignment=assignment)
            for student_to_assignment in students_in_assignment:
                student = student_to_assignment.student
                try:
                    student_to_quiz = StudentToQuiz.objects.get(
                        student=student, quiz=newly_created_quiz)
                except StudentToTopic.DoesNotExist:
                    student_to_quiz = StudentToAssignment(
                        student=student,
                        quiz=newly_created_quiz,
                        grade=0
                    )
                    student_to_quiz.save()

        return successful_create_response(request.data)
    else:
        return colliding_id_response()
```

```
'''
| url: PUT :: <WEBSITE>/api/quiz-question/
| function: Edits a given quiz-question
'''

def put(self, request, format=None, pk=None):
    if pk is None:
        return missing_id_response()
    else:
        try:
            result = self.model.objects.get(pk=pk)
        except self.model.DoesNotExist:
            return object_not_found_response()

        serializer = self.serializer_class(result, data=request.data)

        if not serializer.is_valid():
            return invalid_serializer_response(serializer.errors)

        serializer.save()
        return successful_edit_response(serializer.data)
```

Figure 5: Quiz Question Sample

The various api_views for the Quiz Questions object.

```
// Retrieves a list of all students enrolled in the topic
getStudents() {
  const profile = JSON.parse(localStorage.getItem('profile'));
  axios
    .get(
      `${API_URL}/student/topics/${this.data.topic.id}`, { headers: { Authorization: `Bearer ${profile.auth.profile.id_token}` } }
    )
    .then(studentResponse => {
      this.studentToTopics = studentResponse.data.result;
      for(let i = 0; i < this.studentToTopics.length; i++) {
        this.students[i] = this.studentToTopics[i].student;
      }
    })
    .catch(error => {
      console.log(error);
    })
    // Parse the students into a form that can be read by the dropdown list
    .finally(() => {
      for(let i = 0; i < this.students.length; i++) {
        this.studentCollection[i] = {value: this.students[i].id, text: this.students[i].first_name + ' ' + this.students[i].last_name};
      }
      this.loaded=true;
    });
},
```

Figure 6: Javascript Sample Code

This is a frontend axios method that retrieves a list of all students enrolled in a given topic.

```
<div v-else-if="state == 'TakingQuiz' && this.loaded">
    <h3> {{currentQuestion.question_text}} </h3>
    <div v-for="choice in currentQuestion.choices" v-bind:key="choice.id">
      <input type="radio" :id="choice.id" :value="choice.id" v-model="selection">
      <label :for="choice.id">{{choice.text}}</label>
      <br>
    </div>
    <h5>Question {{currentQuestion.index+1}} of {{numQuizQuestions}}</h5>
    <div>
        <sui-button v-on:click="state = 'StartScreen'">Exit Quiz</sui-button>
        <sui-button id=dynamic-button v-on:click="dynamicButtonClick">{{dynamicButtonState}}</sui-button>
    </div>
    <div v-if="dynamicButtonState == 'Next'">
        <h2> {{submissionFeedback}} </h2>
        <h4> Current grade: {{quizGrade}} </h4>
    </div>
</div>
<div v-else-if="state == 'QuizComplete'">
    <h3> Quiz Complete </h3>
    <h4> Final grade: {{quizGrade}} </h4>
    <sui-button v-on:click="exitQuiz">Exit Quiz</sui-button>
</div>
</div>
</template>
```

Figure 7: HTML Sample Code

HTML for the quiz frontend display.

```
//Gets random questions based on the quiz pool
//The algorithm is O(n), with n being the number of questions.
//TODO: Constants are really bad, consider optimizing if needed.
getQuestions() {
  const profile = JSON.parse(localStorage.getItem('profile'));
  axios
    .get(
      `${API_URL}/quiz-questions/?quiz=${this.quizPK}`, { headers: { Authorization: `Bearer ${profile.auth.profile.id_token}` }})
    )
    .then(response => {
      var questions = {
        0: [], //multiple_choice
        1: [], //free_response
        2: [], //all_that_apply
        3: [], //parsons
      };
      for(let i = 0; i < response.data.result.length; i++) {
        var question = response.data.result[i];
        questions[question.question_type].push(question);
      }

      questions[0] = shuffle(questions[0]);
      questions[1] = shuffle(questions[1]);
      questions[2] = shuffle(questions[2]);
      questions[3] = shuffle(questions[3]);

      this.quiz = [];
      for (let i = 0; i < Math.min(questions[0].length, this.quizPool['multiple_choice']); i++) {
        questions[0][i]['question_parameters'] = JSON.parse(questions[0][i]['question_parameters']);
        this.quiz.push(questions[0][i]);
      }
      for (let i = 0; i < Math.min(questions[1].length, this.quizPool['free_response']); i++) {
        questions[1][i]['question_parameters'] = JSON.parse(questions[1][i]['question_parameters']);
        this.quiz.push(questions[1][i]);
      }
      for (let i = 0; i < Math.min(questions[2].length, this.quizPool['all_that_apply']); i++) {
        questions[2][i]['question_parameters'] = JSON.parse(questions[2][i]['question_parameters']);
        this.quiz.push(questions[2][i]);
      }
      for (let i = 0; i < Math.min(questions[3].length, this.quizPool['parsons']); i++) {
        questions[3][i]['question_parameters'] = JSON.parse(questions[3][i]['question_parameters']);
        this.quiz.push(questions[3][i]);
      }
      this.quiz = shuffle(this.quiz);
    })
    .catch(error => {
      console.log(error);
      this.loadError = true;
    })
    .finally(() => {
      this.loaded=true;
    });
},
```

Figure 8: Frontend Sample Code

Frontend method for retrieving quiz questions.

3.4 Sample Tests

By having substantial unit tests that cover most, if not all, of the product, we ensure that
the implemented features work in the desired circumstance and can handle undesired
circumstances. In addition, tests also help when modifying sections of code by verifying that
they still work properly after a change. Testing services, such as Travis-CI, are useful in this
regard because they ensure only builds that complete all tests are deployed, and thus work as
expected. If there are any issues with the build, the development team is notified and can quickly
fix them, since the team would only have to look at the failing tests to determine the issue.

```python
'''
                                                            Get
    _____

'''
def test_get_students1(self):    # Get All (no pk value)
    dp = {"id_token":"12345"}
    request = self.client.get(path='/api/students/', format='json', data=dp)
    self.assertEqual(200, request.status_code)

def test_get_student2(self):    # Get student with pk value
    student1 = Student.objects.create(
        email="first4last4@gmail.com", first_name="first4", last_name="last4")
    request = self.client.get(path='/api/students/' +
                    str(student1.pk), format='json')
    self.assertEqual(200, request.status_code)

def test_get_students3(self):

    request = self.client.get(path='/api/students/', format='json')
    self.assertEqual(200, request.status_code)
```

Figure 9: GET Student Test Samples

Above, these are three sample tests that test the GET functionality of the student view.
These ensure that, when the proper API call is made, the backend returns the correct information
as well as a status code of 200, which means a success.

```python
def test_topic_cascade3(self):
    self.studentToAssignment1.grade = 90
    self.studentToAssignment2.grade = 90
    self.studentToAssignment1.save()
    self.studentToAssignment2.save()

    student_to_topic = StudentToTopic.objects.get(student=self.student,topic=self.topic)

    self.assertEqual(student_to_topic.competency, 2) # 2 is competent

def test_topic_cascade4(self):
    self.studentToAssignment1.grade = 80
    self.studentToAssignment2.grade = 80
    self.studentToAssignment1.save()
    self.studentToAssignment2.save()

    student_to_topic = StudentToTopic.objects.get(student=self.student,topic=self.topic)

    self.assertEqual(student_to_topic.competency, 1) # 2 is some competentcy

def test_topic_cascade5(self):
    self.studentToAssignment1.grade = 60
    self.studentToAssignment2.grade = 60
    self.studentToAssignment1.save()
    self.studentToAssignment2.save()

    student_to_topic = StudentToTopic.objects.get(student=self.student,topic=self.topic)

    self.assertEqual(student_to_topic.competency, 0) # 0 is not competent
```

Figure 10: Topic Cascading Test Samples

Above, these are tests that cover assignment grade cascading and the mapping of

assignment grades to competency in a topic. The tests assign grades to two assignments in a

topic and ensures that the mapping of grades to competency functions as expected - achieving

high grades on the assignment leads to mastery of the topic, while low grades lead to lower

levels of competency..

3.5 Code Coverage

We are using Django's built in coverage report package to track our python code

coverage. To set up the code coverage report, follow these instructions

20

- sudo docker-compose up

Then, open a new terminal and cd into the src directory and type the following

- sudo docker exec -it backend bash

- coverage run --source='.' manage.py test

- coverage report

The report displays each file that we have written code for, their total lines covered from tests, and total lines available for coverage. At the end of the report, all covered lines and total line counts are summed up in the final coverage report. The following image provides an example to these statistics.

```
.......................................................
-------------------------------------------------------
Ran 121 tests in 3.237s

OK
Destroying test database for alias 'default'...
root@a152690c3be4:/code# coverage report
Name                                           Stmts   Miss  Cover
--------------------------------------------------------------
spt/__init__.py                                    0      0   100%
spt/settings.py                                   31      0   100%
spt/urls.py                                       25      0   100%
sptApp/__init__.py                                 0      0   100%
sptApp/admin.py                                   24      0   100%
sptApp/api_views.py                             1018    440    57%
sptApp/apps.py                                     5      0   100%
sptApp/auth.py                                    64     17    73%
sptApp/cascade_grades.py                          64     20    69%
sptApp/exampleExternalJSON.py                     26      6    77%
sptApp/external.py                                14      0   100%
sptApp/migrations/0001_initial.py                 13      0   100%
sptApp/migrations/0002_auto_20200222_0623.py       4      0   100%
sptApp/migrations/__init__.py                      0      0   100%
sptApp/permissions.py                             22      7    68%
sptApp/responses.py                               34      7    79%
sptApp/serializers.py                             91      0   100%
sptApp/signals.py                                 14      0   100%
sptApp/tests.py                                  781      4    99%
--------------------------------------------------------------
TOTAL                                           2230    501    78%
root@a152690c3be4:/code# 
```

Figure 11: Code Coverage Report Sample

3.6 Installation Instructions

Requirements

- Unix based OS (Linux, MacOS)
  - The install instructions were written for Ubuntu 18.04 LTS

21

- 4GB of ram
  - 8GB is recommended, although we have used as little as 2GB for development and 3GB for production deployments
  - In general, the application takes around 330 MB to run properly, but requires more ram when building the docker containers

Setup

- Clone the repo
- Download Docker and docker-compose
- Setup Google OAuth from the Google API Console
- To install locally for development, follow the Development Install Instructions
- To install on a publically accessible production server, follow the Production Install Instructions

Installing Docker CE and Docker-Compose

1. Install Docker (for ubuntu)

   sudo apt update

   sudo apt install docker.io

2. Test the installation of Docker
   - Run sudo docker run hello-world in the terminal
   - You will see this window on successful install

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figure 12: Docker Install Sample Screen

3. Install Docker-Compose
   ○ Download the latest version
     ■ Run sudo curl -L "https://github.com/docker/compose/releases/download/1.25.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
   ○ Apply permissions
     ■ Run sudo chmod +x /usr/local/bin/docker-compose
   ○ Create a symbolic link
     ■ Run sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
   ○ Test the install
     ■ Run docker-compose --version
     ■ On success you will see an output similar to

```
ubuntu@ip-172-31-21-226:~$ docker-compose --version
docker-compose version 1.23.2, build 1110ad01
```

Figure 13: Docker Compose Sample Output

4. Configure Docker to run on boot
   ○ Run sudo systemctl enable docker in the terminal
5. Configure Docker to run as non-admin
   ○ Run sudo usermod -aG docker ${USER}
   ○ Logout and log back in

For OAuth in development, use the domain spt-acas.com and add it to your /etc/hosts file so that it directs to localhost (instructions in development install instructions). For OAuth in production, use the domain that your server is accessible at.

1. Navigate to the Google API Console
   ○ Login using your gmail
2. Create a new project by clicking on the box next to *Google APIs* then selecting *new project*
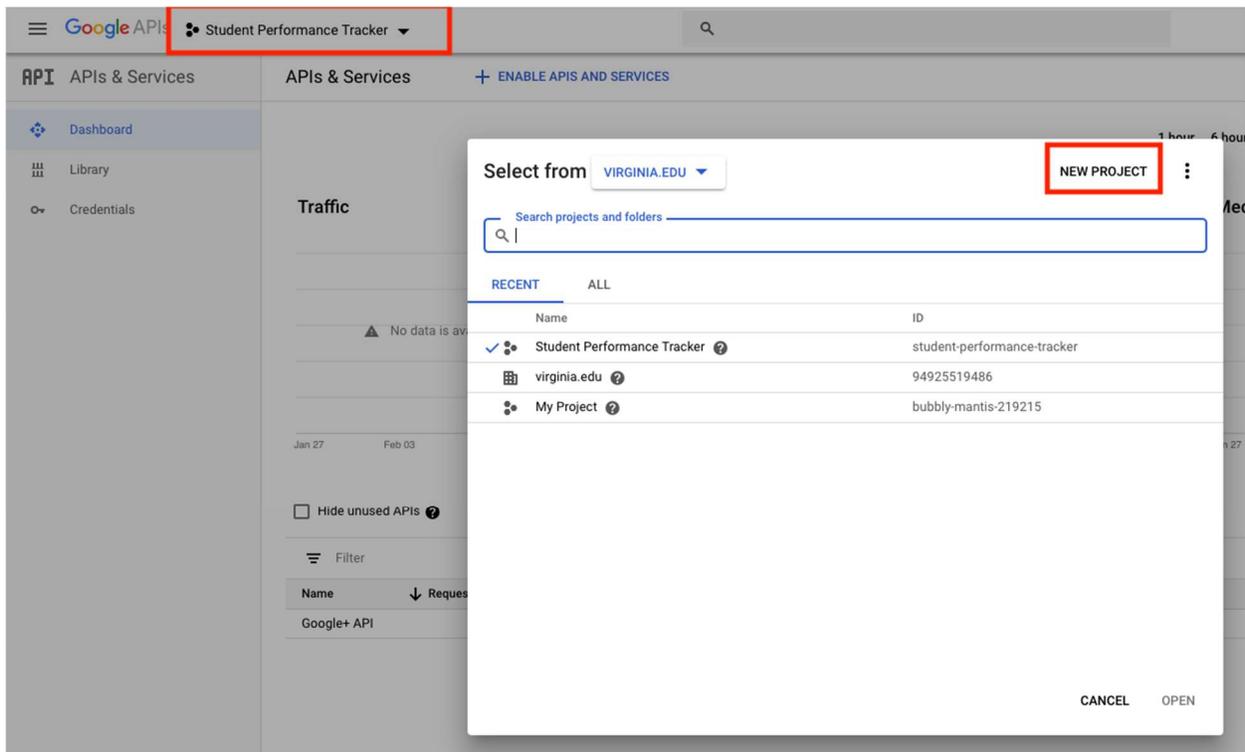

Figure 14: Google API Project Creation

3. Name your new project and click create

Figure 15: Google API Project Name
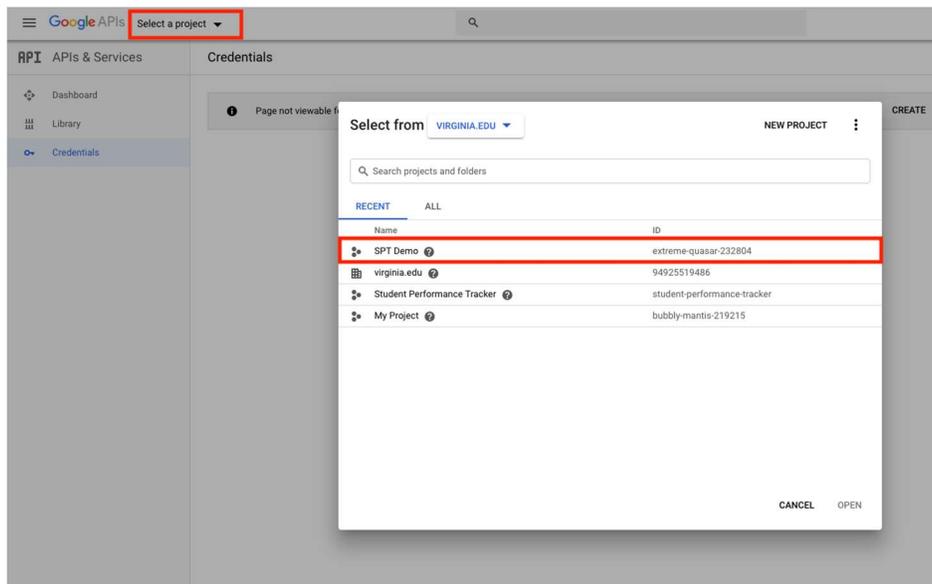
4. Select the new project

Figure 16: Google API Selection

5. Select the *OAuth Consent Screen* tab on the left menu

- Select *External*.

Figure 17: Google OAuth Consent Screen

- Give the application a name
- Add the domain to the authorized domains list.
- Click save
7. Select the *Credentials* tab
    - Click on *Create Credentials*
        - Select *OAuth client ID*
        - Select *Web Application*
            - Name it *SPT* or another name if desired

Figure 18: Google OAuth Client ID Creation

- Add your domain to the *Authorized JavaScript Origins* and the *Redirect URLs*
  - If using the production deployment, use https, otherwise use http
- Click *Save*

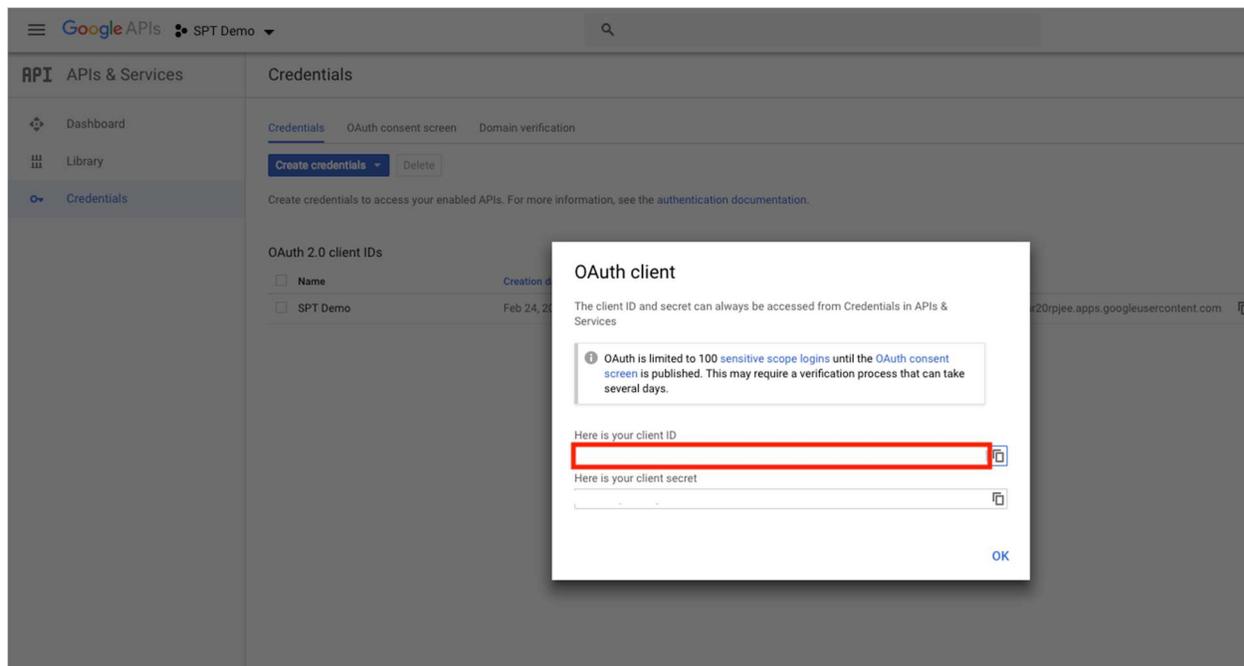8. You will be given a pop-up dialog, copy the *Client ID*

Figure 19: Google OAuth Client ID Editing

- Open the project in an IDE
  - navigate to Auth.js
    - src/frontend/src/vuex/modules/Auth.js
    - Edit the show line to include your Client ID

```
Auth.js                                  ●

1   import api from '../../api';
2   import store from '../index';
3   import axios from 'axios';
4   import { API_URL } from '@/constants/';
5   import { DEBUG_TOKEN } from '@/constants/';
6   //import { IN_PROD } from '@/constants/';
7
8   const state = {
9     signedIn: false,
10    profile: null,
11  };
12
13  const actions = {
14    // Contact the google API
15    initGapi({ commit }) {
16      return new Promise((resolve, reject) => {
17        gapi.load('auth2', {
18          callback: () => {
19            gapi.auth2
20              .init({
21                client_id:
22    '<paste client id here>',
23                })
24              .then(() => {
25                resolve();
26              });
27          },
28        });
29      });
30    },
```

Figure 20: Google OAuth Client ID in the Backend

- ■ navigate to (../src/backend/sptApp/auth.py)
    - ● Edit the SERVER_CLIENT_ID to also match your Client ID

29

Development mode installation instructions

- To enable Google oauth locally in develop mode for sign-up and sign-in, add the following line to your /etc/hosts file (on linux or mac) 127.0.0.1 spt-acas.com
- Navigate to src and run docker-compose up
  - Note: you may have to use sudo when executing docker-compose commands
- Load the debug users into the database. While docker-compose is up, execute the following command in another terminal:

docker exec backend python3 manage.py loaddata debug_users.json

- Create an admin account with the following command:

docker exec -it backend python3 manage.py createsuperuser

The application is available at spt-acas.com:8080. The api is routed to spt-acas:8000. The django admin page is at spt-acas.com:8000/admin

The command docker-compose down will stop the containers but not always delete the database. If you wish to delete the database, execute docker-compose down --volumes

Note: Accessing the application from spt-acas.com is done to allow oauth to work locally since Google oauth requires accessing the application from authorized top level domains. Accessing from spt-acas.com is not required for logging into the debug users. You can access the application from localhost:8080, but oauth will not work for regular sign in and account creation.

Running tests

Tests can be executed with the following commands inside the folder src:

./test-backend.sh

./test-frontend.sh

You may need to add executable permission to execute these scripts: chmod +x *.sh. Note: The application must not be running before running tests.

Production installation instructions

- Note: The following commands must be ran on the production server itself which is accessible by a public domain on the internet
- Set DOMAIN in src/config/deployment_vars to the domain that you will be deploying on
- Setting the EMAIL variable will associate the certificates with your email, which is recommended by letsencrypt
- Inside the src folder, give executable permissions to .sh files

  chmod +x *.sh

- Apply your configuration

  ./apply_deployment_vars.sh

- Build the production files

  docker-compose -f docker-compose.prod.yml build

- If you wish to apply a different configuration, you must first run

  ./reset_deployment_vars.sh

  before applying the new configuration. The build will need to be ran again.
  - Optionally, you can reset the files with git checkout *. Warning: This will lose all local changes to the src files.
- Collect certificates

  sudo ./init-letsencrypt.sh

  - IMPORTANT: You must execute this command on a public server accessible at the domain specified in /src/config/deployment_vars
  - Port 80 must be open and not in use by another program
- Start the server

  docker-compose -f docker-compose.prod.yml up

4. Results

The Student Performance Tracker solves many of the problems that our client was facing. It now allows professors to create and administer automatically graded quizzes with multiple-choice and Parson's problem questions, along with manually graded quizzes with free-response and coding questions. In addition, the previous team's system had API endpoints linking to the backend that were not secure. This meant that anyone who knew the proper secret URL extensions could access the backend, including student grades. These endpoints have been fixed, requiring proper authentication to access now. Professors can now designate teaching assistants that can access and modify student grades, manually grade and regrade quizzes, and help with setting up course topic nodes.

Our group has discussed a testing date with the customer and plan to test our system with him in early April 2020. We will use this to test each feature and collect more feedback from the customer.

5. Conclusions

At its most distilled, the problem which our application seeks to solve is a need for an application through which a Professor can structure a decentralized teaching experience with a directed, acyclical graph of learning topics. Our application not only meets that need, but also exceeds it with the potential for high performance at scale and high potential for adaptation to future needs.

In developing this application, we encountered a number of other problems, especially regarding how best to handle a new codebase and a lack of experience developing with a new framework. Throughout our development experience, we have not only adapted to and overcome these unique challenges, but we have continued to output a usable and modular product, meeting a standard which exceeds those of the former codebase, especially with regards to security. The team has learned various means by which to overcome the issues of unforeseen problems and learning curves, and we have leveraged thoughtful work distribution, regular check-in meetings (both in-person and remote), and small levels of specialization within the group to intelligently continue to move forward.

6. Future Work

As a most immediate need, we seek to test out the new system as quickly as possible, in case there are minor flaws in the design or grading process. Additionally, this application sets a foundation for more pedagogical research regarding ways in which the decentralized learning experience can be helpful. With a modular and scalable application such as this, we can investigate which types of subjects are most viable and which types of students are most advantaged in this system. An application like this also allows for research into which teaching methods and styles are most suitable to a decentralized educational experience, which may assist professors in structuring their classes in the future.

This application has always been designed with improvement and modularity in mind. As a result, future work could also include more simplified efforts, such as expanding the list of features within the application, improving the user interface, and developing a well-documented API such that other researchers can easily modify and use the code which we have written.

This technical report might also serve as a piece of research regarding how best to handle larger amounts of turnover in continuous development of a project. In our particular case, each member of the development team had left, and a completely new docket of workers began to learn about and dissect the code base. It is possible that such a phenomenon happens in the industry as well, and by noting the particular struggles that we as a team faced, we can learn how to lighten the burden on future teams.

7. References

McGrawHill. [n. d.]. Overview of ALEKS. Retrieved from
https://www.aleks.com/about_aleks/overview

McGrawHill. [n. d.]. What is ALEKS. Retrieved from https://www.aleks.com/about_aleks

Pearson. [n. d.]. MyLab and Mastering. Retrieved from
https://www.pearsonmylabandmastering.com/northame
rica/educators/features/index.html