CCA - Academic Tracker

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

Christian Kinzer
Spring, 2020.

Technical Project Team Members
Preston Troxell

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Signature [illegible] Date 5/5/2020
Christian Kinzer

Approved Ahmed Ibrahim Date 5/8/2020
Dr. Ahmed Ibrahim, Department of Computer Science

Table of Contents

Abstract

Modern technology has allowed us to collect and analyze data regarding educational performance of students and teachers. We were approached by Kimberly Moore, the executive director of the Community Christian Academy, to design and implement a system which would allow her to store and analyze students' scores on standardized Stanford 10 tests. Our team originally consisted of six members; however, four of them dropped the capstone course in the beginning of the second semester. We used a Django web framework to provide an online interface that Kimberly would be comfortable using. This interface would allow her to create profiles, add test scores, and produce relevant graphs to analyze performance over time for each student and teacher.

While we were able to produce a minimum viable product with only two team members, this project was rife with struggle. The lessons we learned while working on this project were more about the general process of developing software than technicalities regarding how to build the web interface. Our team did not communicate effectively before the other four students dropped, and the product was very over-engineered. There was little foresight in the decision to use Django, and after our teammates dropped, we realized this product never needed to be web-accessible. At that point, we considered scrapping the product and starting over; however, we decided we were too far along in the timeframe to do so. Technically, in working with Django, we learned how to build a database powered website from scratch, which, while being a valuable skill, did not truly help us meet Kimberly's needs. We hope that at the very least, this paper highlights the importance of fully understanding a problem before attempting to solve it and how we were able to help a small school track student progress inexpensively.

List of Figures

1. Introduction

We were drawn to this project from the start as it was an opportunity for us to involve ourselves in the local Charlottesville community, further our development skills, and provide a service and product for a small organization that had no other options for solving their particular problem. Imagining that our work would lead to improved education in students across many demographics, some with low income, special needs, and/or English as a second language, strongly motivated us to provide our best work for Kimberly and the Academy.

Along the way, we learned invaluable lessons about teamwork, communication, and software development. Some of these lessons had to be learned through failures while others were born of our successes. It is our hope that in this paper, we can communicate those lessons and provide sufficient background such that other developers could read our paper and understand how to further their skills as a member of a team.

1.1 Problem Statement

The Community Christian Academy is a private, affordable, non-denominational Christian school serving the Charlottesville/Albemarle area. The Academy consists of an elementary (Kindergarten - 5th grade) and middle school (6th - 9th grade). Every year, students in each grade take Stanford 10 tests (Onyeberechi, n.d.) to evaluate their yearly performance. It is vital for these test scores to be easily analyzed so that the Academy can identify and help struggling students. Prior to this academic year, Kimberly would manually enter test scores into an Excel spreadsheet with very limited analysis capability. The spreadsheet Kimberly used would simply highlight low scores in red and high scores in green, with no notion of comparing student performance over time. While this system allowed her to see individual tests where students struggled, it did not provide a very complete picture of student performance. In addition, there was no information regarding the teachers of the students, which would be similarly beneficial in the case of an underperforming teacher. While Kimberly did not take issue with the time required to manually enter the scores, analyzing student performance over time would require her to view data spread across multiple spreadsheets and was very cumbersome.

Our system provided Kimberly with automatic analysis given data input. The database driven website allowed us to create automated, dynamic graphs for both student and teacher performance over their time at the Academy. This primary functionality, amongst other minor features, provides Kimberly with the ability to make better informed decisions on how the Academy should operate.

1.2 Contributions

In the end, we decided not to host our web interface on a server, as Kimberly was not prepared to pay for the cost of hosting. Fortunately, Kimberly was comfortable using our system locally. We were able to create a local web based application allowing her to create student and teacher profiles, enter test data, and view raw data and statistics in addition to graphs of performance over time. The rest of this thesis is organized in four sections. In Section 2, related work is presented. Section 3 shows our approach to address the aforementioned problem statement, our web-based application, as well as the system design. The results of our work are discussed in Section 4. Finally, Section 5 concludes this thesis, with Section 6 discussing future work regarding this problem.

2. Related Work

Many other systems exist that provide similar functionality to our product. There are a multitude of generic systems, such as PowerSchool, that are offered at a cost to a private school such as the Academy (Powerschool, 2020). The Academy does not have sufficient funding to use a service that is not free. Free generic systems also exist (MasteryConnect, 2020); however, they do not meet the specific requirements of CCA. These systems require instructors to write their own assessments and students to take these assessments online (Formative, 2020). Kimberly was looking for a system to track standardized test scores sent to her on paper.

In addition to generic solutions, Kimberly has also employed the UVA Capstone program in past years in an attempt to find a custom written solution. These previous attempts have had many bugs and in certain instances failed to meet Kimberly's requirements. The goal of our team was to improve upon these past attempts and to create a working system with no maintenance costs.

3. System Design

The goal of our system was to create an interface for one user (Kimberly) to input data on students, teachers, and tests to receive statistics and dynamic, automated graphs as output. These statistics and graphs should show a larger picture of the students' performance over time, primarily by filtering the students by various demographics. We used Django (2020), a Python based web development tool to create the interface. We chose Django because all of our original team members had prior experience using it and felt comfortable extending that experience into a new problem. We also decided that a web interface would allow Kimberly to use the product frictionlessly, provided that the layout and design were logical and clean.

3.1 System Requirements

When working with a customer, it is important to develop system requirements with them so that both parties have the same understanding of what constitutes a finished product. We separated our requirements into minimum requirements (features that would constitute a minimum viable product), desired requirements (features that we should implement if we have the resources), and optional requirements (features that could be implemented if everything else is finished).

Minimum Requirements

As a USER, I should be able to….

● View student profiles (fields described in admin) through either a dropdown and/or searching by name (first, last, or a combination)

● View data (student or test) in a table that can be sorted by column headings

(ascending/descending order based on a selected attribute)

● Compare sets of students given some attribute(s)

● Filter test data by subject, year, grade level

● View a teacher's categories & associated students and their corresponding data

● Create student profiles consisting of: first & last name, gender (male/female), ethnicity, English as a second language (true/false), single parent family (true/false), special needs (yes/no), low income (yes/no)

● Create test consisting of: grade level, sub-tests (categories), date of test, number of students tested, link to the categories model, number of questions, mean number correct, mean scaled score, national PR-S of the Mean National NCE, mean national NCE, at/above the 50th national PR, Median Grade Equivalent

● Create test data for each student on a given test, including: number of questions correct, scaled score, national PR-S, national NCE, grade equivalent, and a flag of whether they are under grade level

● Edit existing student profile data

● Delete student profiles

● Create teacher profiles consisting of: first & last name and categories they teach

● Add comments to a student's profile

Desired Requirements

As a USER, I should be able to….

● Filter displayed table data based on specific attribute (single parent family, special needs, etc.)

● Modify test fields to: a) swap order of categories, b) add additional categories, c) edit existing categories, d) delete categories

● Stack multiple filters together for a student or category (e.g. filtering by both ethnicity & english as a second language)

● Visualize student's progress through grade levels on specific categories - connected scatter plot

● Compare student's score to average performance on the selected test (broken up based on category) - stacked bar plot

● Dynamically change filters on visualizations

Optional Requirements

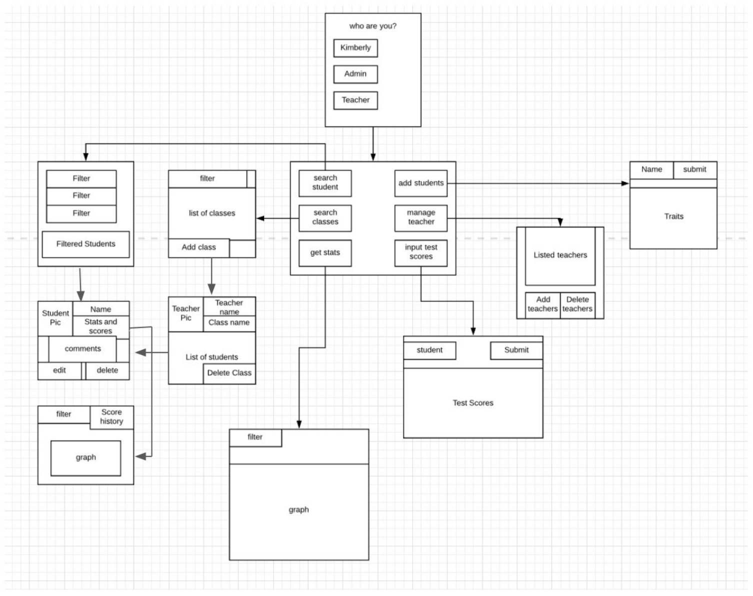
As a USER, I should be able to….

● Download student data/visualizations as a .xlsx or .csv

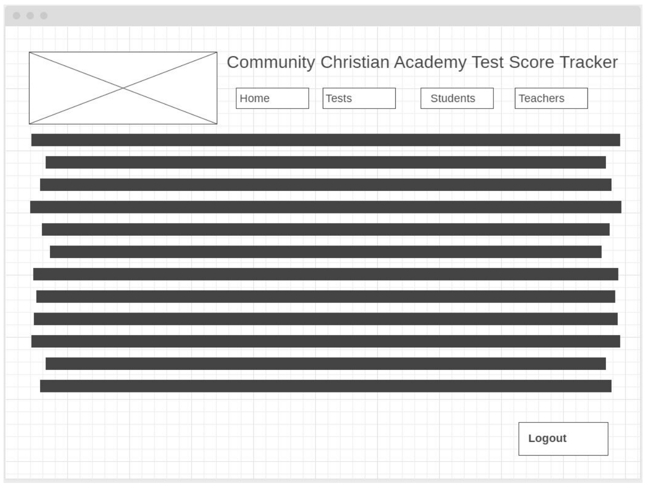
● Upload a scan of student data to automatically be parsed into a new student profile/test

3.2 Wireframes

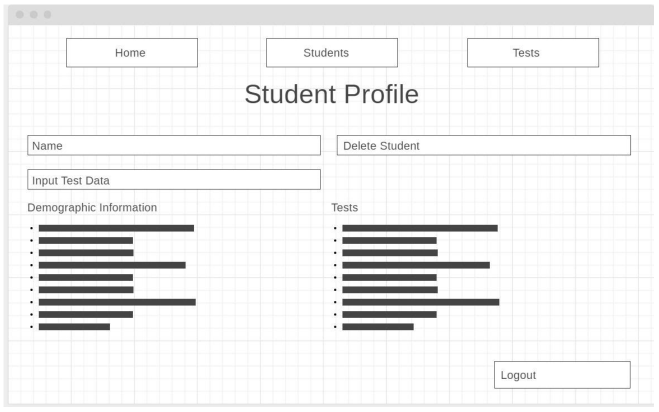
Wireframes ensure that a customer has an understanding of the layout and flow of the product before the code is fleshed out. Wireframes also provide a frame of reference for the programmers as they develop the product.



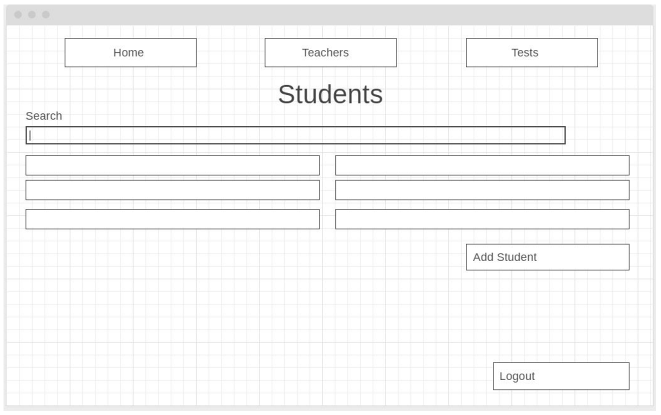
1. Navigation wireframe.



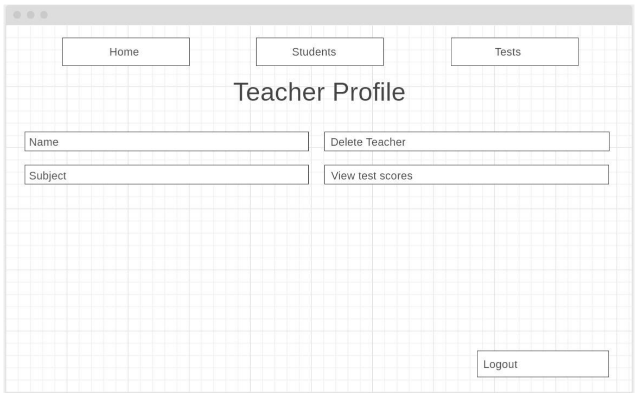
2. Homepage wireframe.



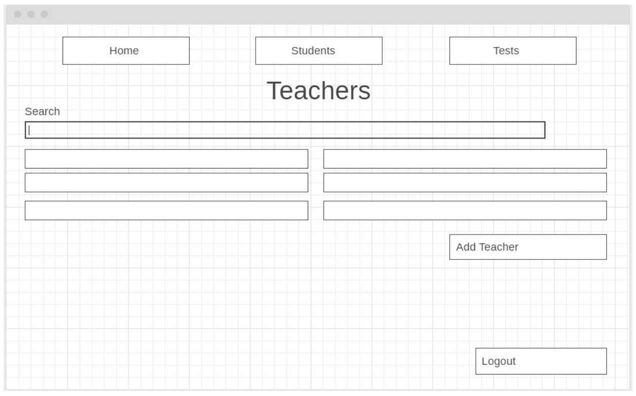
3. Student Profile wireframe.



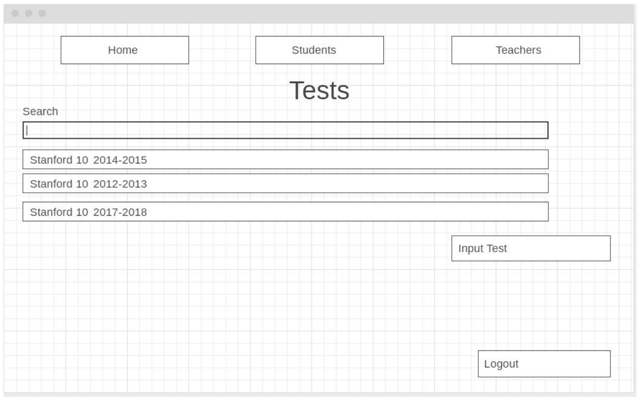
4. Student landing wireframe.



5. Teacher Profile wireframe.

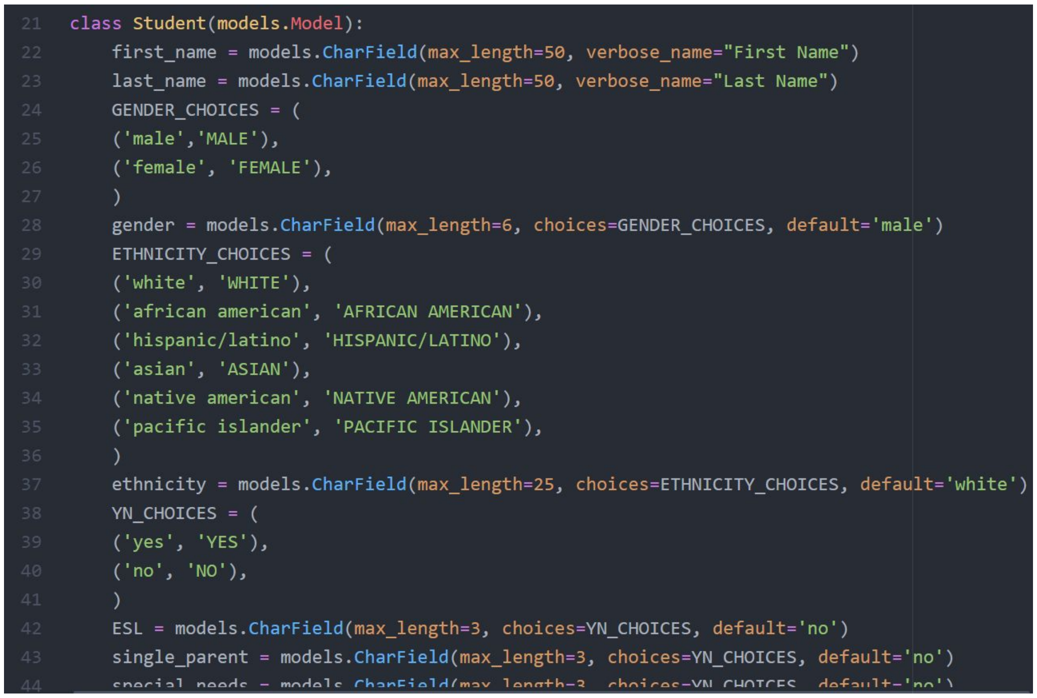


6. Teacher landing wireframe.



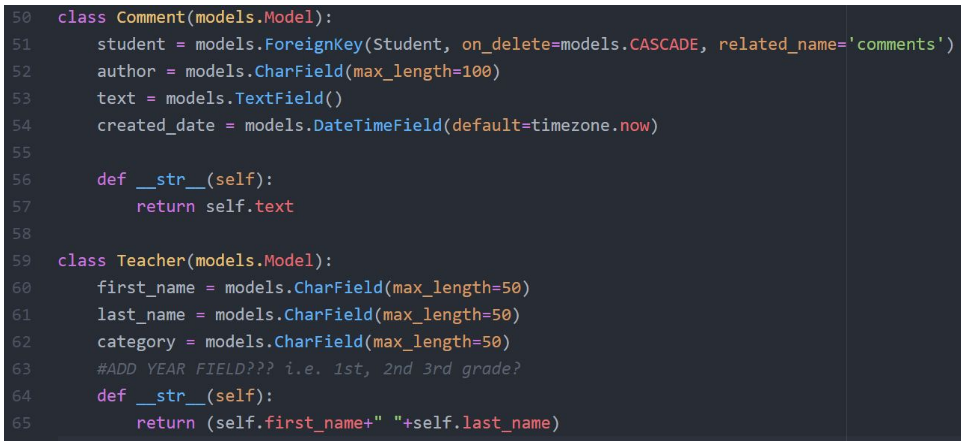
7. Tests landing wireframe.

3.3 Sample Code



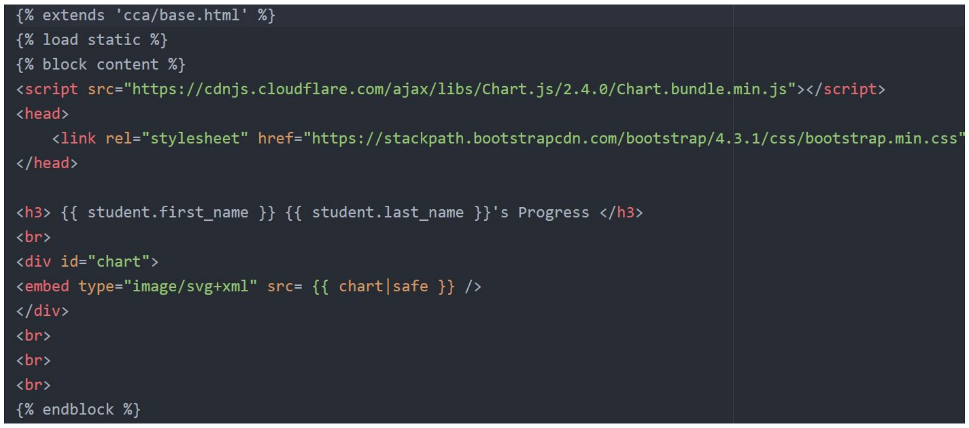
```python
class Student(models.Model):
    first_name = models.CharField(max_length=50, verbose_name="First Name")
    last_name = models.CharField(max_length=50, verbose_name="Last Name")
    GENDER_CHOICES = (
    ('male','MALE'),
    ('female', 'FEMALE'),
    )
    gender = models.CharField(max_length=6, choices=GENDER_CHOICES, default='male')
    ETHNICITY_CHOICES = (
    ('white', 'WHITE'),
    ('african american', 'AFRICAN AMERICAN'),
    ('hispanic/latino', 'HISPANIC/LATINO'),
    ('asian', 'ASIAN'),
    ('native american', 'NATIVE AMERICAN'),
    ('pacific islander', 'PACIFIC ISLANDER'),
    )
    ethnicity = models.CharField(max_length=25, choices=ETHNICITY_CHOICES, default='white')
    YN_CHOICES = (
    ('yes', 'YES'),
    ('no', 'NO'),
    )
    ESL = models.CharField(max_length=3, choices=YN_CHOICES, default='no')
    single_parent = models.CharField(max_length=3, choices=YN_CHOICES, default='no')
    special_needs = models.CharField(max_length=3, choices=YN_CHOICES, default='no')
```

8. Student model.



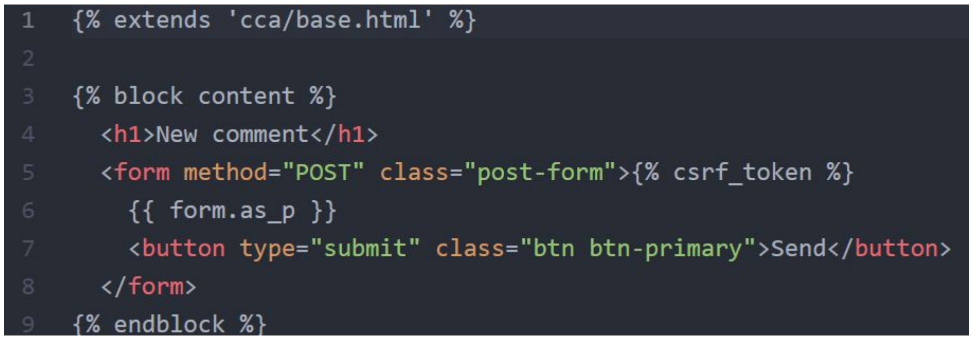
```python
class Comment(models.Model):
    student = models.ForeignKey(Student, on_delete=models.CASCADE, related_name='comments')
    author = models.CharField(max_length=100)
    text = models.TextField()
    created_date = models.DateTimeField(default=timezone.now)

    def __str__(self):
        return self.text

class Teacher(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    category = models.CharField(max_length=50)
    #ADD YEAR FIELD??? i.e. 1st, 2nd 3rd grade?
    def __str__(self):
        return (self.first_name+" "+self.last_name)
```

9. Teacher and comment models.



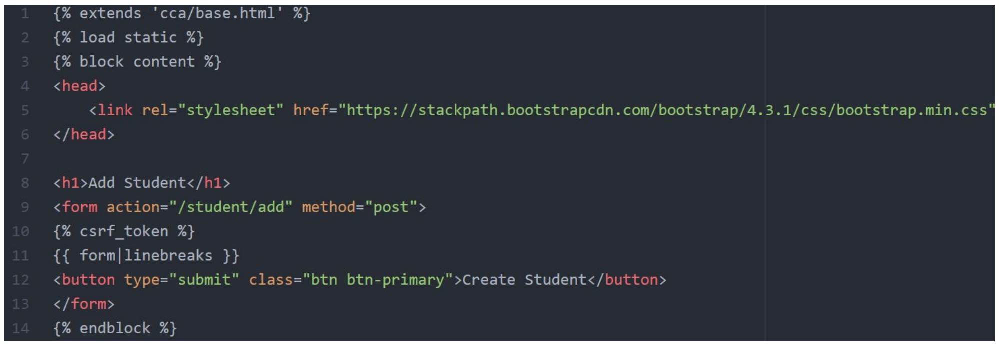
```
{% extends 'cca/base.html' %}
{% load static %}
{% block content %}
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.4.0/Chart.bundle.min.js"></script>
<head>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
</head>

<h3> {{ student.first_name }} {{ student.last_name }}'s Progress </h3>
<br>
<div id="chart">
<embed type="image/svg+xml" src= {{ chart|safe }} />
</div>
<br>
<br>
<br>
{% endblock %}
```

10. View for student progress.



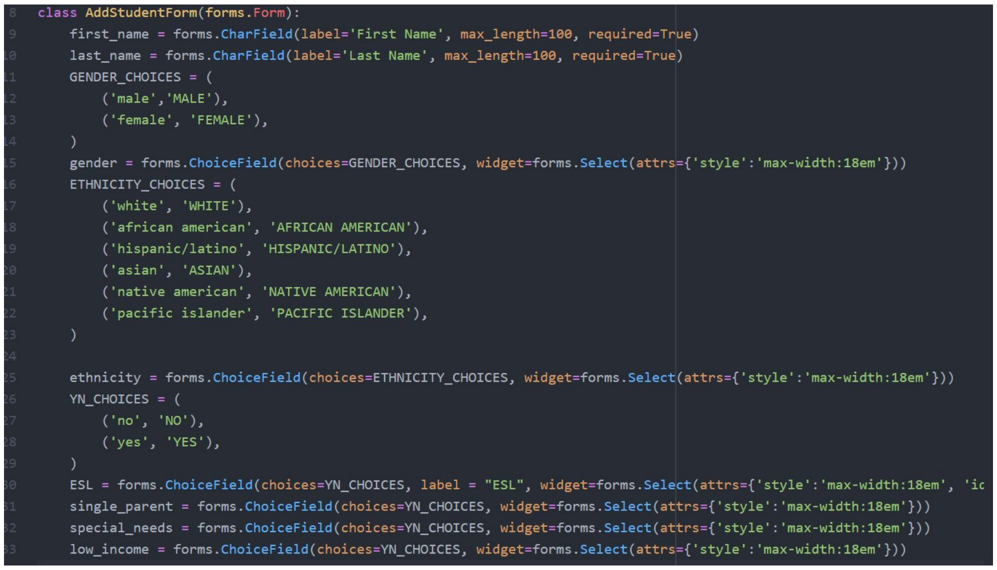
```
{% extends 'cca/base.html' %}

{% block content %}
  <h1>New comment</h1>
  <form method="POST" class="post-form">{% csrf_token %}
    {{ form.as_p }}
    <button type="submit" class="btn btn-primary">Send</button>
  </form>
{% endblock %}
```

11. View for adding a comment to a student's profile.



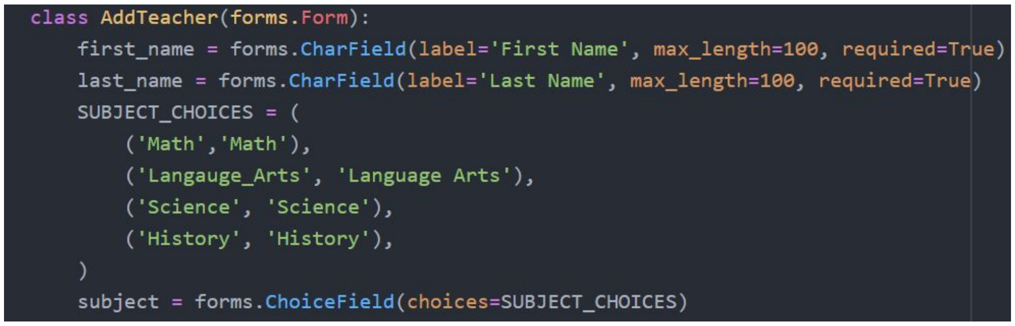
```
{% extends 'cca/base.html' %}
{% load static %}
{% block content %}
<head>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
</head>

<h1>Add Student</h1>
<form action="/student/add" method="post">
{% csrf_token %}
{{ form|linebreaks }}
<button type="submit" class="btn btn-primary">Create Student</button>
</form>
{% endblock %}
```

12. View for adding a new student to the database.



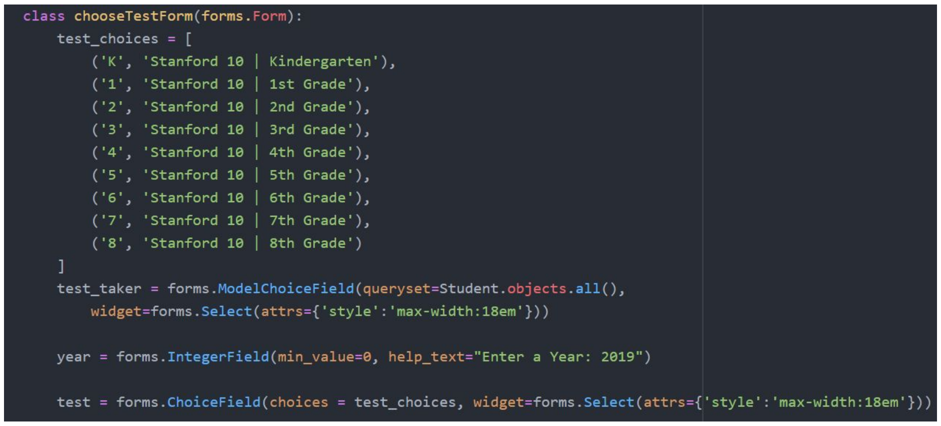
```python
class AddStudentForm(forms.Form):
    first_name = forms.CharField(label='First Name', max_length=100, required=True)
    last_name = forms.CharField(label='Last Name', max_length=100, required=True)
    GENDER_CHOICES = (
        ('male','MALE'),
        ('female', 'FEMALE'),
    )
    gender = forms.ChoiceField(choices=GENDER_CHOICES, widget=forms.Select(attrs={'style':'max-width:18em'}))
    ETHNICITY_CHOICES = (
        ('white', 'WHITE'),
        ('african american', 'AFRICAN AMERICAN'),
        ('hispanic/latino', 'HISPANIC/LATINO'),
        ('asian', 'ASIAN'),
        ('native american', 'NATIVE AMERICAN'),
        ('pacific islander', 'PACIFIC ISLANDER'),
    )

    ethnicity = forms.ChoiceField(choices=ETHNICITY_CHOICES, widget=forms.Select(attrs={'style':'max-width:18em'}))
    YN_CHOICES = (
        ('no', 'NO'),
        ('yes', 'YES'),
    )
    ESL = forms.ChoiceField(choices=YN_CHOICES, label = "ESL", widget=forms.Select(attrs={'style':'max-width:18em', 'ic
    single_parent = forms.ChoiceField(choices=YN_CHOICES, widget=forms.Select(attrs={'style':'max-width:18em'}))
    special_needs = forms.ChoiceField(choices=YN_CHOICES, widget=forms.Select(attrs={'style':'max-width:18em'}))
    low_income = forms.ChoiceField(choices=YN_CHOICES, widget=forms.Select(attrs={'style':'max-width:18em'}))
```

13. Form for adding a student to the database.



```python
class AddTeacher(forms.Form):
    first_name = forms.CharField(label='First Name', max_length=100, required=True)
    last_name = forms.CharField(label='Last Name', max_length=100, required=True)
    SUBJECT_CHOICES = (
        ('Math','Math'),
        ('Langauge_Arts', 'Language Arts'),
        ('Science', 'Science'),
        ('History', 'History'),
    )
    subject = forms.ChoiceField(choices=SUBJECT_CHOICES)
```

14. Form for adding a teacher to the database.

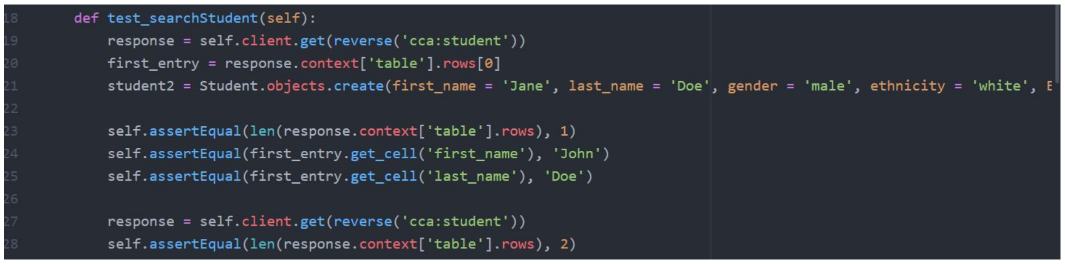


```python
class chooseTestForm(forms.Form):
    test_choices = [
        ('K', 'Stanford 10 | Kindergarten'),
        ('1', 'Stanford 10 | 1st Grade'),
        ('2', 'Stanford 10 | 2nd Grade'),
        ('3', 'Stanford 10 | 3rd Grade'),
        ('4', 'Stanford 10 | 4th Grade'),
        ('5', 'Stanford 10 | 5th Grade'),
        ('6', 'Stanford 10 | 6th Grade'),
        ('7', 'Stanford 10 | 7th Grade'),
        ('8', 'Stanford 10 | 8th Grade')
    ]
    test_taker = forms.ModelChoiceField(queryset=Student.objects.all(),
        widget=forms.Select(attrs={'style':'max-width:18em'}))

    year = forms.IntegerField(min_value=0, help_text="Enter a Year: 2019")

    test = forms.ChoiceField(choices = test_choices, widget=forms.Select(attrs={'style':'max-width:18em'}))
```

15. Form for choosing a test, taker, and year to add to the database.

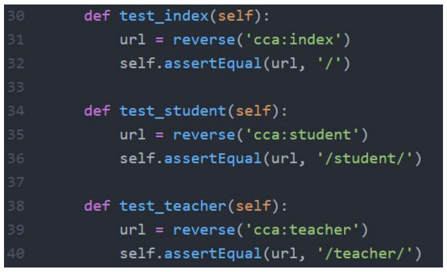
3.4 Sample Tests

When writing software, unit tests are important because they test features automatically as you develop them. This ensures your code is behaving as you expect and makes it easier to detect bugs in the software.



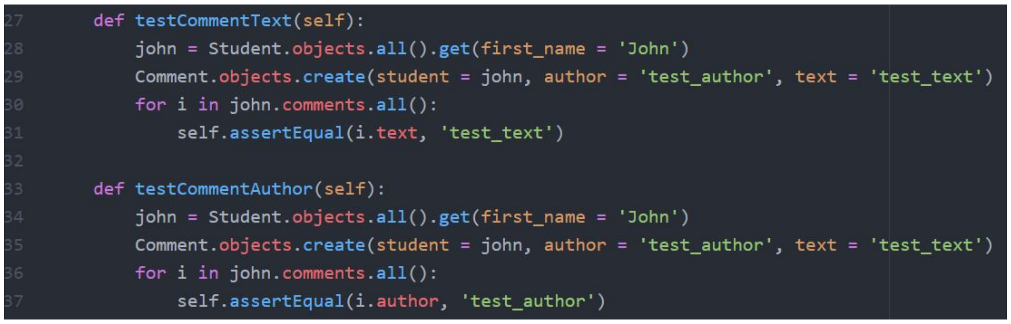
```python
    def test_searchStudent(self):
        response = self.client.get(reverse('cca:student'))
        first_entry = response.context['table'].rows[0]
        student2 = Student.objects.create(first_name = 'Jane', last_name = 'Doe', gender = 'male', ethnicity = 'white', E

        self.assertEqual(len(response.context['table'].rows), 1)
        self.assertEqual(first_entry.get_cell('first_name'), 'John')
        self.assertEqual(first_entry.get_cell('last_name'), 'Doe')

        response = self.client.get(reverse('cca:student'))
        self.assertEqual(len(response.context['table'].rows), 2)
```

16. Test for our student search.



```python
    def test_index(self):
        url = reverse('cca:index')
        self.assertEqual(url, '/')

    def test_student(self):
        url = reverse('cca:student')
        self.assertEqual(url, '/student/')

    def test_teacher(self):
        url = reverse('cca:teacher')
        self.assertEqual(url, '/teacher/')
```

17. Test for various URL mappings.



```python
    def testCommentText(self):
        john = Student.objects.all().get(first_name = 'John')
        Comment.objects.create(student = john, author = 'test_author', text = 'test_text')
        for i in john.comments.all():
            self.assertEqual(i.text, 'test_text')

    def testCommentAuthor(self):
        john = Student.objects.all().get(first_name = 'John')
        Comment.objects.create(student = john, author = 'test_author', text = 'test_text')
        for i in john.comments.all():
            self.assertEqual(i.author, 'test_author')
```

18. Tests for adding a comment to a student.

3.5 Code Coverage

To test our code coverage, we used coverage.py. We integrated it with Django according to the instructions listed at https://docs.djangoproject.com/en/2.0/topics/testing/advanced/#integration-with-coverage-py. Our final code coverage was 89%, indicating that when our test suite ran all of our unit tests, 89% of the codebase was executed and tested.

3.6 Installation Instructions

This is an installation guide for the CCA Student Assessment Application, the targeted OS being Windows 10. To get this running you will need to complete the following steps:

1. Install Python
2. Code Set-Up
3. Install Dependencies
4. Running the Server

On the first instance of completing this Installation Instructions manual, the end result should be the user being able to launch the server. After the first instance of running through this manual, subsequent attempts to run the server should go directly to Step 4) Running the Server - Steps 1,2, and 3 are one-time installations.

1) INSTALLING PYTHON:

- To install python go to https://www.python.org/downloads/windows/ and download Python 3.7.6.
- Click “Download Windows x86-64 executable installer”.

Run the downloaded .exe file and accept all default options. Python should be installed. To verify the installation of python, open a Windows PowerShell window by searching in the search bar in the bottom left of the default Windows User Interface. When you open the PowerShell, run the command "python --version" by typing into the power shell and pressing the Enter key.

2) CODE SET-UP:

- Download Source Code
    - To install the source code navigate to this link (https://github.com/uva-cp-1920/CCA) and download CCA.zip.
- On your computer, navigate to the folder that contains this file (by default this will be the

Downloads folder) - right click this file and click "Extract All…" and then click "Extract". The files should be now visible in a folder called CCA.

3) INSTALL DEPENDENCIES:

- Navigate to the CCA folder that you just extracted and Shift + Right Click in the file explorer. In

the dropdown menu that pops up, click the option "Open PowerShell Window here".

- To install the dependencies, run the following commands in the PowerShell:

```
- pip install Django==2.1.2
- pip install django_tables2
- pip install django_crispy_forms
- pip install pygal
- pip install django-jquery
- pip install django-bootstrap4
```

- pip install numpy

- To run these commands, simply type them in the PowerShell window and hit the Enter key after each line. Once these packages are all installed, you will be ready to run the code!
- Open File Explorer and navigate to the extracted CCA folder. Double-click to open the folder, then double-click to open the folder labelled src. Next, double-click to open the mysite folder.
- While in this folder, highlight the current path above the listed folders and then type 'cmd' in the highlighted upper bar and click the Enter key.
- A command prompt should open up showing your current path. The first time ever opening this command prompt: (One-Time Instructions)
    - Type 'python manage.py makemigrations' and press the Enter key and wait for this process to finish.
    - Next, type 'python manage.py migrate' and press the Enter key and wait for this process to finish.

4) RUNNING THE SERVER:

- Open File Explorer and navigate to the extracted CCA folder. Double-click to open the folder, then double-click to open the folder labelled src. Next, double-click to open the mysite folder.
- While in this folder, highlight the current path above the listed folders and then type 'cmd' in the highlighted upper bar and click the Enter key.
- To run the server, type 'python manage.py runserver' and press the Enter key. To access

the website, visit http://localhost:8000.

- When done with the current session, you can simply close the command prompt by clicking the ‘X’ icon on the top right and the server will save any modified information. On the next instance of running the server you may skip the ‘python manage.py makemigrations’ and ‘python manage.py migrate’ steps.

4. Results

This project presented a variety of problems to be solved, mainly dealing with data input, presentation and analysis. In order to provide the administrators at the academy with valuable insights, we first had to find a method of porting the data into the system, since it was provided on paper. Optimally, this problem would be solved by scraping scanned versions of these documents for the test scores, but this line of thinking was quickly abandoned as it became clear that such a task would be beyond the scope of the project, and because there was too much variability in the formatting of the data. Ultimately, we pivoted to a manual data input format, which is more cumbersome, but is sufficient to solve this issue.

The next problem we were faced with was presenting the data in a way that is digestible, and that provides key information about how particular students are doing and why. The first step we took to solve this problem was to simply display students' personal information and test scores over time on their profile page, which allows school administrators to see how each student is doing relative to their grade level at a glance. We then provided further insights in the form of graphs, which can show how a student is trending over time, and whether or not they are keeping up with their peers. Additionally, we built in a filtering system which allows for analysis of certain cross-sections of students, such as those who speak English as a second language, or those with learning difficulties. Finally, test scores can be analyzed by teacher, providing an additional source of information that can indicate why scores rise or fall in a given year.

Ultimately, the customer will use this system to enhance their ability to focus school resources on students who are not receiving enough attention. Although a large portion of time will have to be invested up front to manually input each student's test scores for each year, there is no net loss of time, since they were already required to do this. This product will save the user time in analysis, because once the data is in the system the user is instantly capable of engaging with the built in analytical tools, whereas before they would have had to create them on their own. Additionally, these tools will be much easier to use, since filters can be applied with just the click of a button, and can compare sets of data that might have been difficult to identify using other tools. The end user has not yet used the product so measured data is unavailable at this point, but we estimate that over time this product could save hours of time spent in trying to manually manipulate and display data.

5. Conclusions

In conclusion, this project taught the group what it means to build a fully realized web application from the ground up, and the many challenges that may arise along the way. It provided us with extensive hands on experience with the Django framework, which will be invaluable moving forward should any of us pursue a career in web development. It also showed us that completing a project like this takes much more than basic coding skills--it requires frequent communication amongst team members and with the various stakeholders who oversee production. Furthermore, ensuring that the final product is viable and meets the specifications of a non-technical party, which we found in Kimberly Moore, can be challenging; determining the specifications for a product based on a limited understanding of computer science often left the group to make assumptions about what was to be delivered, rather than looking to a concrete set of requirements.

The main obstacle this group encountered throughout the course of the last two semesters has been in communication. This is made evident by the fact that two thirds of the group dropped the course in the final two months of development. Although these members never made clear to the remaining team members their reasons for dropping the course, we can say with confidence that it was due to a severe lack of communication between group members.

This experience yielded valuable insights on how a group project as large as this one should be managed. First, it is absolutely essential for members to communicate early and often, in order to ensure that the workload is evenly distributed, and that no members are left behind as the product gets off the ground. From the start, our group was split into two groups who knew each other before the project's inception, and the fact that this gap was never bridged led to almost all of the challenges described in this document. Furthermore, the challenges we faced make it clear that it is crucial to have a single, dedicated project manager, whose job it is to distribute tasks at the start of each sprint, ensure that each member is progressing, and help any member who is struggling, whether that be with technical problems or interpersonal issues within the team.

6. Future Work

There are many potential improvements to the system the team created. The user interface of the web module was very simplistic, and very little color exists beyond the generated graphs. A possible extension of this project would be to host the system on a web server, such that many computers and users could interact with the system. Given that our product is hosted locally, we created a product that is really only meant to be used by one person. A school such as CCA might find value in a similar system in which teachers as well as administrators could interact with the system under different user accounts.

Another area of potential improvement is data input. Data entry has to be performed manually on our system, given the nature of the data being handled (tests of various formats). If this project had a larger scope, we could have tried to implement some kind of system to scan in the paper results and automatically upload testing data to the database (PDFMiner, 2014). We anticipate that this would be a major improvement on the overall convenience of the system and would probably be worth pursuing with more time and research.

7. References

Django (2020). Django documentation. https://docs.djangoproject.com/en/3.0/

Formative (2020). Home. https://goformative.com/

MasteryConnect (2020). Home. https://www.masteryconnect.com/

Onyeberechi (n.d.). Characteristics of Students Who Score High on the Stanford Achievement Test. *Seattle Post-Intelligencer*.

PDFMiner (2014, Mar 24). Python PDF parser and analyzer. http://www.unixuser.org/~euske/python/pdfminer/index.html

PowerSchool (2020). Home. https://www.powerschool.com/