Power of Difference Assessment System

A Technical Report submitted to the Department of Computer Science

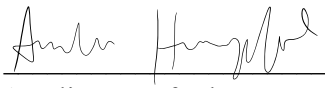
Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

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

Amelia Hampford
Spring, 2020

Technical Project Team Members
Peter Felland
Nuzaba Nuzhat
Sam Shankman
David Xue
Connor Yager
Carl Zhang
William Zheng

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  Date _____
Amelia Hampford

Approved  Date 4/22/2020
Dr. Ahmed Ibrahim, Department of Computer Science

Table of Contents

Abstract

The Power of Difference Assessment (PDA) is an assessment system designed by The Sum, a Charlottesville based non-profit, that helps people learn about their demographic biases and understand how to better communicate with those who differ. Since its conception in 2015, users have been able to take the assessment online at www.powerofdifference.org. But in recent years, various issues related to user friendliness, security, and lack of automation have emerged, which has prompted the owner of the product to request a complete overhaul. Our eight member capstone team was charged with this overhaul, with goals that included fixing the existing issues, adding numerous administrative functionalities, and ensuring scalability to help the PDA grow beyond its foundational years, and into a service that can be used for much broader institutional studies.

The team worked in an agile methodology with two week sprints. Biweekly meetings with the customer provided technical requirements to direct the work. The Django web framework, based on the Python programming language, was used to ease and guide the development process. The system was deployed using Amazon Web Services (AWS) to support many concurrent users. Results include a fully functioning PDA system that lets users register, take the PDA, receive automatically generated results, and set up a consultation with The Sum. It also allows The Sum to have persistent data storage, manage user information, and visualize statistics from assessment results. The customer has already benefited from these functionalities; for example, instead of making the results PDF manually, which took up to five minutes with the old system, the results are now automatically generated and emailed to users, which takes less than a minute and can be done for multiple users simultaneously. Functionalities like this and various others have led the new PDA to be much more usable,

scalable, and secure compared to its predecessor, and establish itself as a valuable tool in The Sum's range of offerings.

List of Figures

1. Introduction

People with demographic differences can struggle to communicate with each other. People often “make decisions based on what [they] already believe” without listening to other perspectives (Headlee, 2015). This makes communication competitive instead of cooperative (Zenger & Folkman, 2016). With poor communication, people become desensitized to what others say (Treasure, 2011). People can see topics in a new light and better understand others when they listen to diverse perspectives (Zenger & Folkman, 2016).

The Sum, led by Elliott Cisneros, is a non-profit located in Charlottesville, VA, which promotes personal growth, skill development, and diversity. The goal of The Sum is to stand in solidarity with all people, no matter their background. Towards this goal, they provide a wide range of services that include workshops, discussions, training sessions, etc. One such offering is the Power of Difference Assessment (PDA), which is based on the Power of Difference Model. Users can access the PDA through institutional access codes, or one time payments. After signing up and answering questions about their demographics, users are asked to respond to 70 statements that range across various sociocultural locations, such as age, race, gender, sexuality, etc., on a likert scale. Once they are done, a report with results is generated and emailed to the participant. The results are categorized across demographics, areas of strength, and areas of growth, and can help reveal people’s biases, and allow them to understand the role they play in combating or sustaining existing socio-economical injustices. Those who take the PDA can meet with a consultant from The Sum to learn about their biases, understand how to better communicate across various demographic differences, and overcome their own limitations to help establish a more “just, joyful life, community and world” (“The Sum”, n.d.).

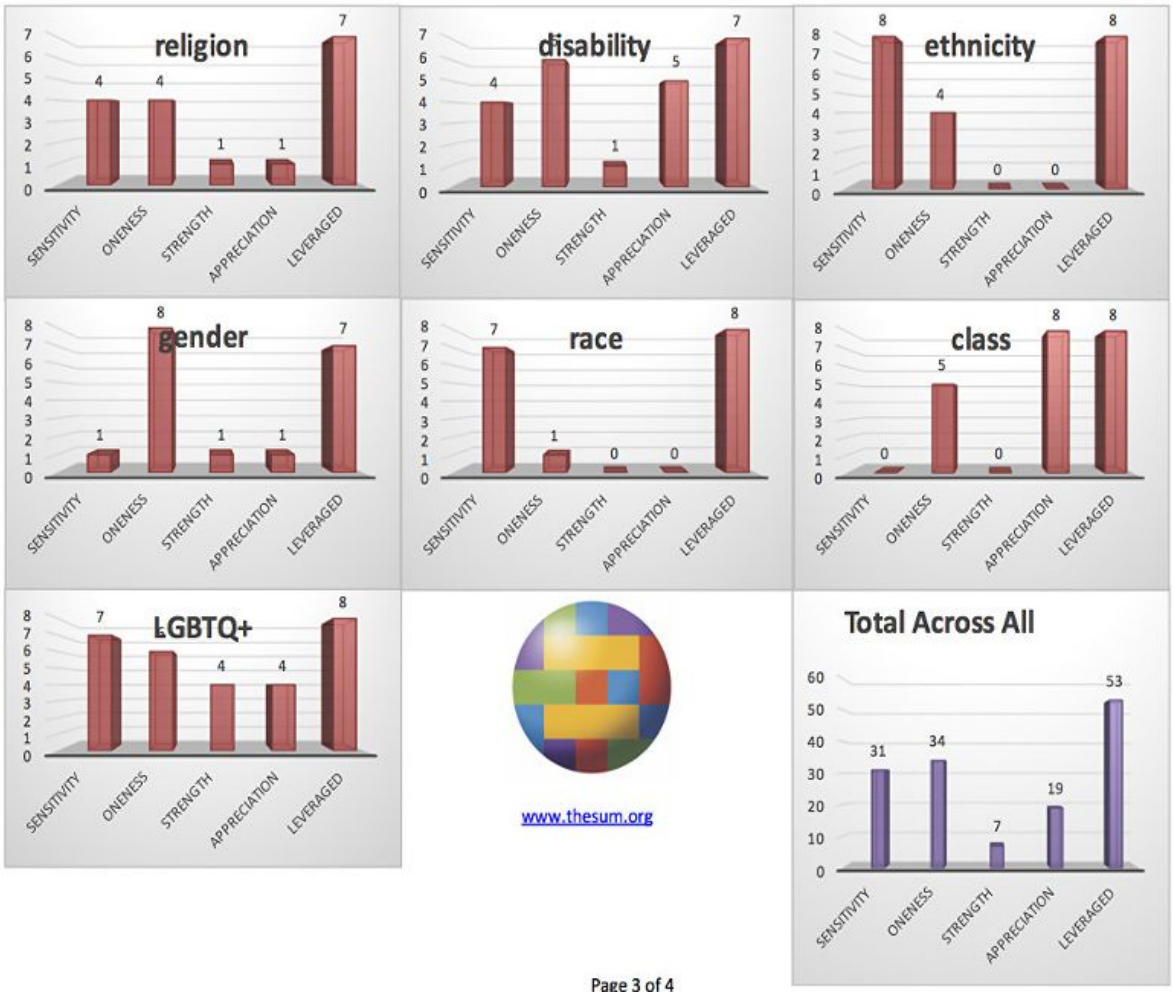


Figure 1: Sample result generated from the old version of the PDA. There are seven sociocultural locations: religion, disability, ethnicity, gender, race, class, and, LGBTQ+, and five power perspectives: sensitivity, oneness, strength, appreciation, and leveraged. To become an individual ready to create positive change in the world, high leveraged scores and nearly zero scores for everything else ("The Sum", n.d.).

1.1 Problem Statement

The Sum already has an online PDA system in place. The current system allows users to take the PDA and schedule a consultation. However, the system is error-prone. The system improperly categorizes results and it requires someone at The Sum to manually generate

reports and email them to users. As part of report generation, categorizations are manually checked and corrected. This makes report generation time consuming and prone to human error. Although manually generating a report only takes a few minutes, the time from PDA completion to reports being emailed to users varies based upon availability at The Sum, and can take up to 24 hours. In addition, reports are only generated one at a time. This methodology is not scalable and cannot support the upcoming study of 1,000 PDA takers. In addition to this, the current system does not detect a difference between assessment versions which helps with record keeping. It is also insecure and allows for URL manipulation so users can manipulate data in the system. And although the system is generating abundant data that can be used to further improve the model, the product owner has no access to this data, making even simple analysis impossible.

The new PDA system will be beneficial by solving these problems. It will allow for automatic results generation to help speed up the workflow of The Sum, thus allowing them to support more users. The new system should also be scalable, also supporting more users and the upcoming study of 1,000 PDA takers. Detecting versions of the PDA helps The Sum to stay organized and provide proper consultations to users. Finally, the added administrative functionality such as viewing user data, adding and assigning consultants, visually analyzing demographics and responses, should make the new system much more useful to The Sum's staff members at various levels, who find no use in the current system.

1.2 Contributions

The new PDA system solves the issues faced by the old system as well as maintaining base functionality so users can take the PDA. We made a web-based PDA system that automatically generates results correctly, creates a PDF, and emails them to users. This takes

less than a minute per PDF and saves time for The Sum. In addition, we set up the system on AWS so it can handle many users simultaneously, allowing users to take the assessment and view their PDF results real time. The increase in scale makes the upcoming study of 1,000 PDA takers possible. We also made the new system detect which version of the PDA is being taken. This lets the system direct users to the proper page to schedule consultations with The Sum based upon the taken PDA version. Finally, we have built-in protections to prevent URL manipulation that could occur, limiting the improper manipulation of data by users.

2. Related Work

Tests like the Myer-Briggs Type Indicator and the DiSC Assessment work to improve communication effectiveness. They focus on personality, while the PDA project focuses on demographics (“MBTI® Basics”, n.d.; “DiSC Overview", n.d.). Project Implicit studies people’s biases and raises bias awareness, but does not help to improve communication across demographics (“Project Implicit”, n.d.). There are other systems similar to the Myer-Briggs Type Indicator and the DiSC Assessment, but none of them focus on the entire combination of demographics, raising bias awareness, and improving communications. The Sum has a system in place, but it is error prone and not user friendly.

Current systems typically focus on personality, while the new PDA system will focus on demographics and biases. The new system fixes the issues with the current PDA system while maintaining all base functionality. The new PDA system provides a system that helps to raise demographic bias awareness. By receiving consultations from using the new system, users can reflect on themselves and learn how to better communicate across demographics.

As for pre-existing technologies with similar functionalities, there are numerous options available; some are free, others are paid. The renowned services include Google Forms, Survey

Monkey, Qualtrics etc. However, none of them include the registration, result generation and administrative capabilities our customer was looking for, which is why we decided to build a custom website that can successfully meet and adapt to his requirements.

3. System Design

The new PDA system should allow users to register with a valid email address. After verification, these users can select which version of the PDA they want to take and pay if needed. After this, users can take the PDA and receive an email with their automatically generated results. When finished, users should be directed to a page where they can schedule a consultation based upon which version of the PDA they took. For The Sum, the new PDA system should allow them to view all the responses, demographics, results, and contact information of users. The system was developed in Python using the Django web framework as it is widely supported with many helpful packages to increase productivity. Since the new PDA system is a web application, HTML, JavaScript, and CSS were used to design the web pages. The code falls under an MIT license which means the code is provided as is without warranty. Anyone can use the code so long as all copies of the code contain a copy of the license (MIT License, n.d.).

3.1 System Requirements

In order to make the new system, requirements had to be gathered from The Sum. Requirements determine what features should be part of the new system and which features should be prioritized. Feature prioritization impacts the development timeline. Requirements help track development progress. Separating the work into requirements allows the team to

determine who works on which features. Most importantly, requirements establish clarity between the capstone project team and The Sum for what is to be built.

Minimum requirements are:

- As an admin, I want users to sign up with a valid email address, name, and phone number.
- As an admin, I want users to undergo email verifications before starting the assessment.
- As an admin, I want to prevent users from going backwards in the assessment once they have started.
- As an admin, I want users to fill out their demographics before taking the assessment.
- As an admin, I want users to answer each and every question of the assessment.
- As an admin, I want users to only view one question at a time.
- As an admin, I want users to answer the questions in order.
- As an admin, I do not want users to be able to change responses to questions.
- As an admin, I want users to be able access the consultation scheduling system once they have finished taking the assessment.

Desired requirements are:

- As an admin, I want the system to correctly categorize the responses to questions.
- As an admin, I want the system to automatically and correctly generate pdf reports.
- As an admin, I want the system to email pdf reports to the users and The Sum automatically.
- As an admin, I do not want users to be able to go back and view previous questions.
- As an admin, I want a paid version of the PDA that includes sliding scale options.
- As an admin, I want an institutional version of the PDA, for which I can generate access codes, and set the number of times an access code can be used.

- As a user, I want to be able to select which version of the assessment I take.
- As an admin, I want to be able to allow users to retake the assessment.
- As an admin, I want the system to be scalable.
- As an admin, I want the system to have persistent data storage.
- As an admin, I want an admin interface to look at and modify data.

Optional requirements are:

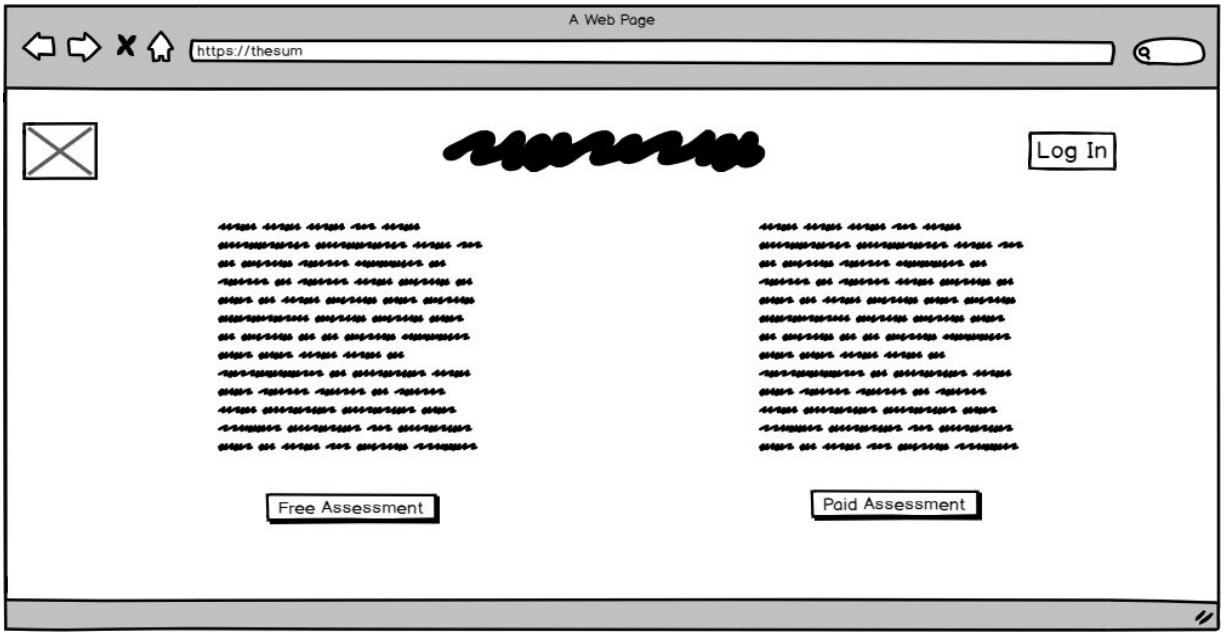
- As a user, I want to be able to take the assessment on mobile devices.
- As an admin, I want the website servers to support https.
- As an admin, I want the ability to reset my password.
- As an admin, I want to be able to create consultant accounts.
- As an admin, I want to be able to assign assessments to consultants.
- As an admin, I want to prevent consultants from modifying data, and viewing data for assessments they are not assigned to.
- As a consultant, I want to be notified when I am assigned a new assessment.
- As a consultant, I want to view all relevant data for the assessments I have been assigned to.
- As an admin, I want to have a class of admins that can only view data.
- As an admin, I want to be able to visualize different aspects of the data, such as demographics breakdown of all the users, response breakdown for all statements, etc.

3.2 Wireframes

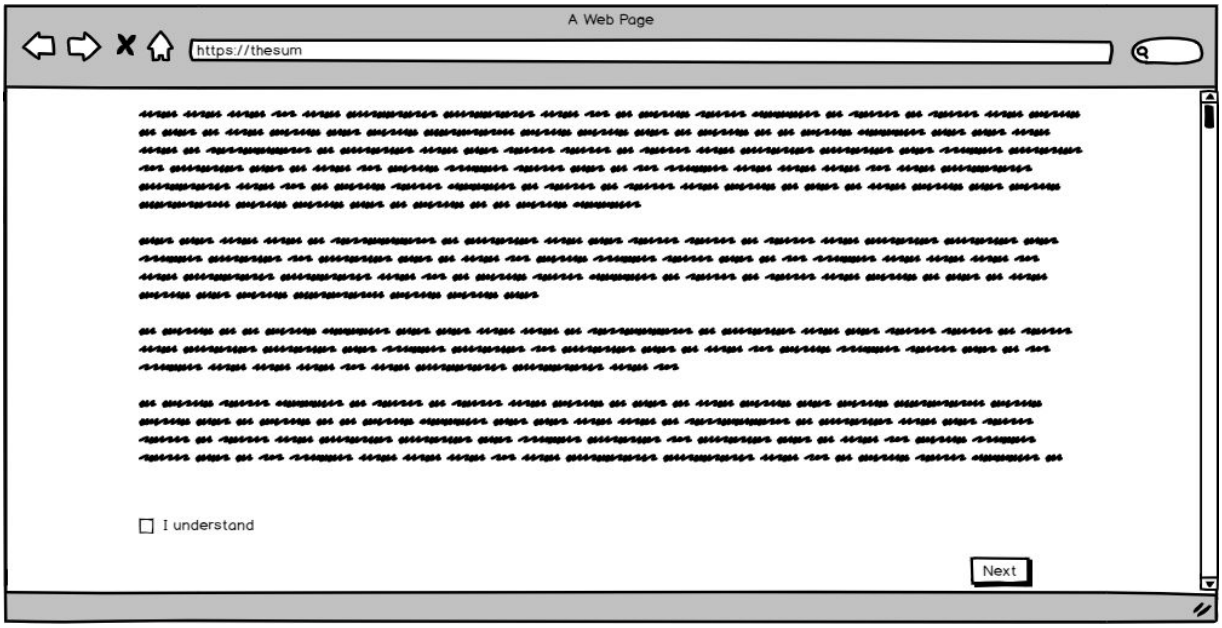
Wireframes are important as they establish structure before development begins. They help to give each page a purpose towards meeting the requirements. Wireframes create a hierarchy of pages and information which allows the team and the customer to think about how

the application should be used. This hierarchy can also help establish which elements are the most important. Thinking about how the application should be used is important as decisions can be made that provide the best user experience. Finally, wireframes help to establish clarity. They allow the development team to understand what features are desired. They also establish clarity about what is to be built between the customer and the developers.

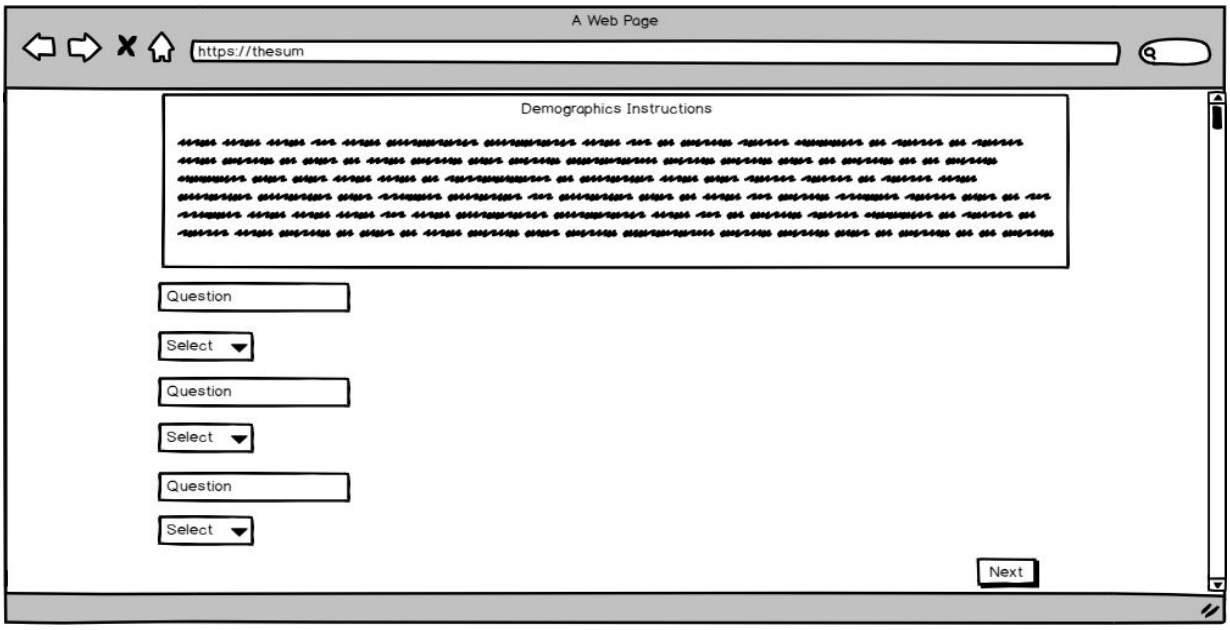
Wireframe 1: The registration page



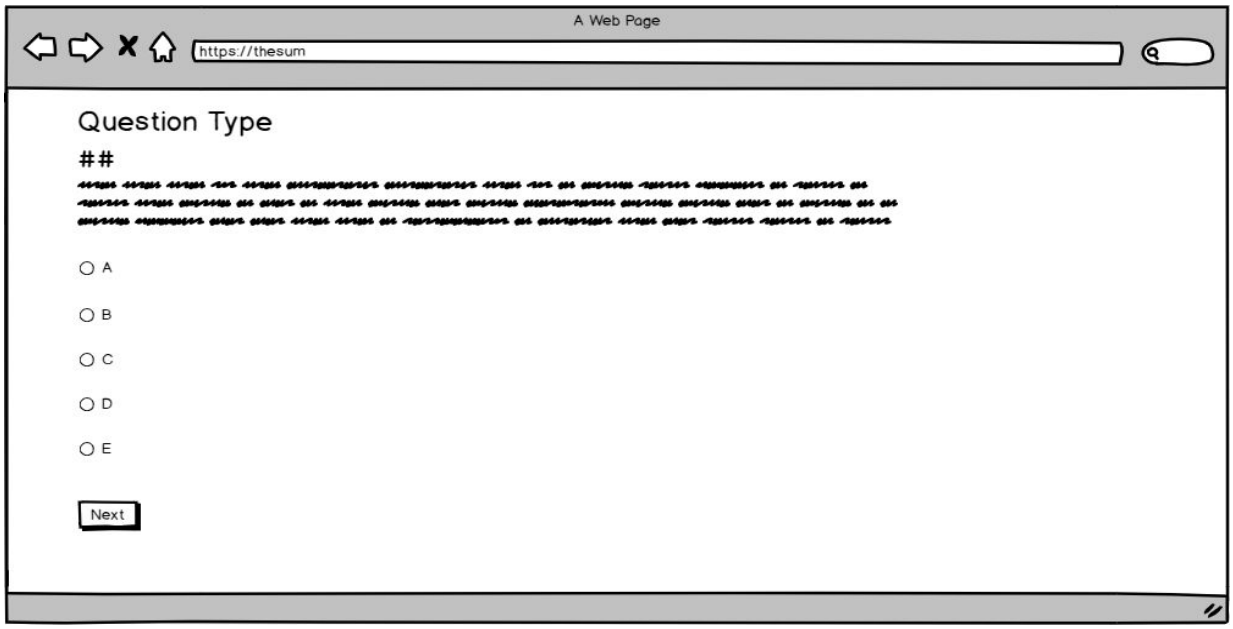
Wireframe 2: Choosing the type of assessment



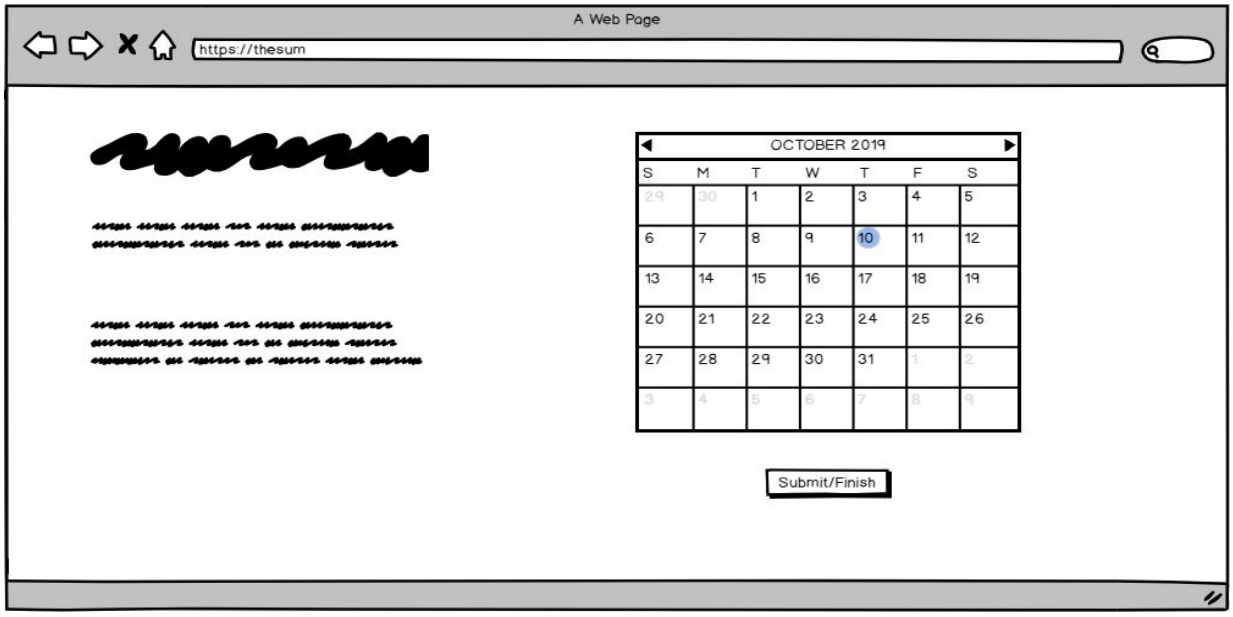
Wireframe 3: PDA instructions



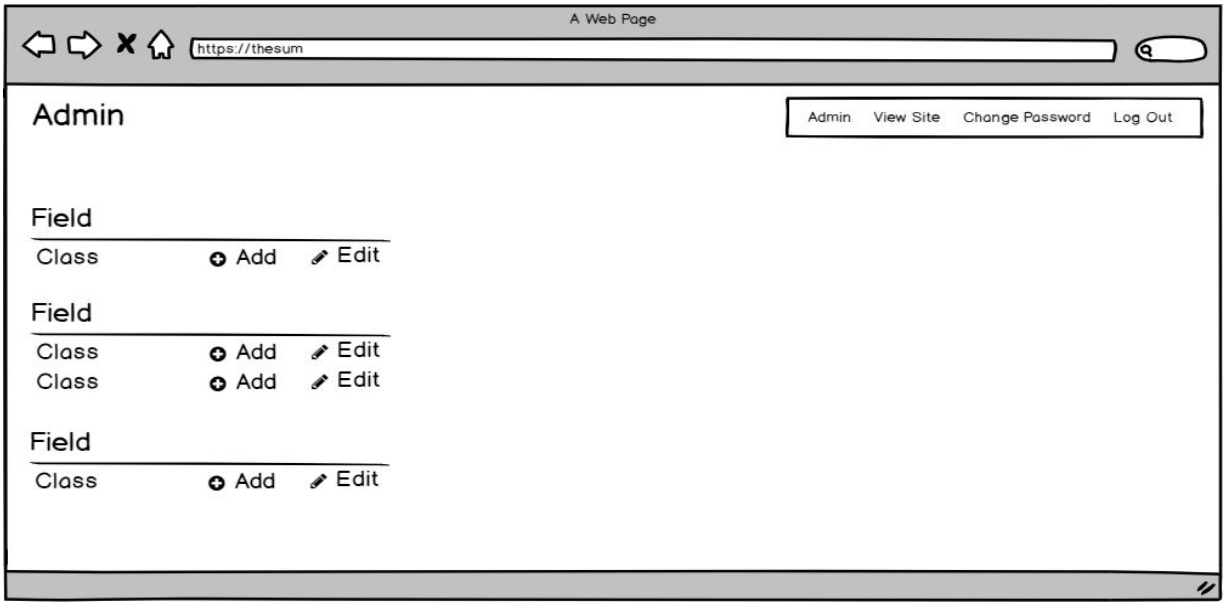
Wireframe 4: PDA demographics questions



Wireframe 5: PDA question



Wireframe 6: PDA consultant scheduling page. Viewed after finishing the assessment.



Wireframe 7: Administrator view of data

3.3 Sample Code

Each piece of sample code will have comments at the beginning of it to describe the functionality or purpose of the code. There will be groups of sample code pieces to show business logic, forms, models (database tables), and HTML pages. The first two pieces of sample code are samples of business logic for taking the assessment.

```
# Make sure the user is on the correct statement
# Load data the statement
# Is this the last question? If so, UI will display different text for
submit button
@incomplete_assessment_required
def question(request, user_id, assessment_id, number):
   last_question = request.session['last_question']
   if number != last_question + 1:
       kwargs = dict(user_id=user_id, assessment_id=assessment_id,
number=last_question + 1)
      return HttpResponseRedirect(reverse('core:question', kwargs=kwargs))
   question = Question.objects.get(number=number)
   next_question = Question.objects.filter(number=number + 1)
   is_last_question = False
   if not next_question:
       is_last_question = True
   context = dict(user_id=user_id,
                  assessment_id=assessment_id,
                  question=question,
                  is_last_question=is_last_question,
                  number=number,
                  form=ResponseForm())
   return render(request, 'core/question.html', context)
```

```
# Check if the access code entered is valid
# If it is, decrement uses (when it does not have infinite uses)
# If it is not valid or out of uses, tell the user
@require_http_methods(["POST"])@require_session_key_absence('access_type')
def verify_access_code(request, user_id):
   if 'access_code' not in request.POST:
       return HttpResponseRedirect(reverse('core:choose_access_type',
kwargs=dict(user_id=user_id)))
   access_codes =
```

```
AccessCode.objects.filter(code=request.POST['access_code'])
   if len(access_codes) > 0:
       access_code = access_codes[0]
       if access_code.uses_left == -1 or access_code.uses_left > 0:
           request.session['access_type'] = ACCESS_TYPE_INST
           if access_code.uses_left != -1:
               access_code.uses_left -= 1
               access_code.save(update_fields=['uses_left'])
           return HttpResponseRedirect(reverse('core:create_assessment',
kwargs=dict(user_id=user_id)))
       else:
           messages.error(request, ACCESS_CODE_NO_USE_LEFT_MESSAGE)
   else:
       messages.error(request, INVALID_ACCESS_CODE_MESSAGE)
   return HttpResponseRedirect(reverse('core:choose_access_type',
kwargs=dict(user_id=user_id)))
```

The next two pieces of sample code are samples of forms that are meant to receive user input.

```
# Form field to check if a phone number is valid
class PhoneField(CharField):
   default_validators = [RegexValidator(regex=phone_regex, message="Phone
number is not a valid US Phone number")]
   widget = TextInput(
       attrs=dict(title="Examples: 1234567890, 123-456-7890, 123.456.7890,
123 456 7890 or (123) 456 7890",
                  pattern="^\(?[0-9]{3}\)?[-.●]?[0-9]{3}[-.●]?[0-9]{4}$",
placeholder="ex. 123-456-7890"))
```

```
# Form for user registration
# Use the phone form field that validated data
# Sample data so users know what they should enter
class UnverifiedUserForm(ModelForm):
   def __init__(self, *args, **kwargs):
      super(UnverifiedUserForm, self).__init__(*args, **kwargs)
       self.fields['email'].widget.attrs['autofocus'] = 'true'
```

```
        self.fields['email'].widget.attrs['placeholder'] = 'ex.
john.doe@example.com'
        self.fields['first_name'].widget.attrs['placeholder'] = 'ex. John'
        self.fields['last_name'].widget.attrs['placeholder'] = 'ex. Doe'

    class Meta:
        model = UnverifiedUser
        fields = ['email', 'first_name', 'last_name', 'phone']
        field_classes = {
            'phone': PhoneField,
            'email': CharField
        }
```

The next two pieces of sample code are samples of models. Models are used to define tables in the database.

```
# an assessment has an access type, a related user, email, date started,
question it should be on, PDF of results, and consultant who can view it
class Assessment(models.Model):
    ACCESS_TYPES = [(ACCESS_TYPE_PAID, 'Paid'), (ACCESS_TYPE_INST,
'Institution')]

    user = ForeignKey(User, on_delete=models.SET_NULL, primary_key=False,
null=True, blank=True)  # many assessments to one user
    access_type = CharField(max_length=4, choices=ACCESS_TYPES,
default='INST')
    email = EmailField()
    date_started = DateField(default=timezone.now)
    last_question = IntegerField(default=0, verbose_name='Last Statement')
    PDF = models.FileField(upload_to='pdfs/', null=True, blank=True,
storage=PrivateMediaStorage())
    consultants = models.ManyToManyField(Consultant, blank=True)
```

```
# responses to statements
# tied to an assessment and question
# have a response value (level of agreement), power perspective, and
sociocultural location in order to categorize the response
class Response(models.Model):
  assessment = ForeignKey(Assessment, on_delete=models.CASCADE,
primary_key=False, null=True, blank=True)
```

```
    question_number = IntegerField(default=0,
                                   verbose_name='Statement Number')  # if we
need to reference a response back to a question.
    RESPONSES = [("strongly agree", "Strongly agree"),
                 ("agree more than disagree", "Agree more than disagree"),
                 ("agree and disagree about the same", "Agree and disagree
about the same"),
                 ("disagree more than agree", "Disagree more than agree"),
                 ("strongly disagree", "Strongly disagree")]
    response = CharField(choices=RESPONSES, max_length=50, default="test")
# stores the choice
    power_perspective = CharField(choices=POWER_PERSPECTIVES, max_length=50)
    sociocultural_location = CharField(choices=SOCIOCULTURAL_LOCATIONS,
max_length=50)  # (race, religion, etc)
```

The last two pieces of sample code are samples of HTML pages that users will see.

```
<!-- Display the question number, title, and have a form where users can
select level of agreement -->
 <!-- Upon submission, go to the question submission code -->
                <h2 class="card-title">Statement {{ question.number }}</h2>
                <h5 class="card-title">{{ question.title }}</h5>
                <form class="card-text col-md-12 mx-auto bg-white p-4
rounded-lg" role="form" method="POST"
                      action="{% url 'core:question_submit' user_id=user_id
assessment_id=assessment_id number=question.number %}">
                    {% csrf_token %}
                    <div class="form-group" role="radiogroup">
                        {% for radio in form.response %}
                            {{ radio.tag }}
                            <label for="{{ radio.id_for_label }}">
                                {{ radio.choice_label }}
                            </label>
                            <br>
                        {% endfor %}
                    </div>
```

```
<!-- part of the HTML code for choosing an access type. This is for the
paid access types with a form related to paypal  -->
<div class="col-md-6" role="gridcell">
```

```
                <div class="card px-0 text-center h-100">
                    <h4 class="card-header font-weight-bold">
                        Paid Access
                    </h4>
                    <div class="card-body d-flex flex-column
justify-content-between">
                        <p class="card-text">If you want to take the PDA and
receive an hour-long results consultation (on
                            the phone or in-person), please click here to
pay through PayPal. The cost is
                            $150.00. </p>
                        {{form.render}}
                    </div>
                </div>
            </div>
```

3.4 Sample Tests

Testing code is important as it lets the team easily find bugs and make sure the system behaves as expected. Instead of finding bugs late in development, tests allow the team to find bugs early and fix them before they become bigger issues. Tests also help to ensure stable code as code will not be put into production unless it passes tests. Finally, tests can help drive development. Writing tests for how code should behave before the code is written allows the team to understand what they should be developing. This also helps test functionality based on what the system should do rather than based on what code was written. The following sample tests have some comments at the start of each test to help describe the purpose of the test.

```
# a test that makes sure only a single email is sent at a time (to a single
user)
   # makes sure the system tried to save the pdf (but will not actually
save it in the test)
@patch('storages.backends.s3boto3.S3Boto3Storage.save')
def test_single_email(self, mock_save):
       naming = "pdfs/" + "tj3vavirginia.edu" + "_" +
str(self.assessment.pk) + "_results.pdf"
       mock_save.return_value = naming
```

```
        session = self.client.session
        session['user_id'] = self.user.pk
        session['assessment_id'] = self.assessment.pk
        session['access_type'] = ACCESS_TYPE_PAID
        session['last_question'] = len(Question.objects.all())
        session.save()
        response = self.client.get(
            reverse('core:score', kwargs=dict(user_id=self.user.pk,
assessment_id=self.assessment.pk)),
            follow=True)
        self.assertEquals(len(mail.outbox), 1)
        mock_save.assert_called_once()  # No file saved!
```

```
# tests appreciation subscore
   # tests scoring of responses
   # the strongly agree puts 4 points into the race appreciation score
section
   def test_score_sub_score_race_appreciation_test(self):
        Response.objects.create(assessment=self.assessment,
question_number=1, response="strongly agree",
                                power_perspective="Appreciation",
sociocultural_location="Race")

self.client.post(reverse('core:score',kwargs=dict(user_id=self.user.pk,
assessment_id=self.assessment.pk)))
        self.score = Score.objects.get(pk=1)
        self.race_score = Race_Score.objects.get(score=self.score)
        self.assertEqual(self.race_score.appreciation, 4)
```

```
# a test to make sure that we do not let the same email register twice
   # create a user, get their data, and try to register with that same data
again
   # expect our message that a user already exists with that email
   def test_email_already_exists(self):
        """A new unverified user is not created and error gets thrown"""
        user = create_user()

self.assertEquals(len(UnverifiedUser.objects.filter(email=user.email)), 0)
        post_body = dict(user.__dict__)
```

```
        response = self.client.post(reverse('core:register'), post_body,
follow=True)
        self.assertRedirects(response, reverse('core:register'))

self.assertEquals(len(UnverifiedUser.objects.filter(email=user.email)), 0)
        messages = list(response.context.get('messages'))
        self.assertEquals(len(messages), 1)
        message = messages[0]
        self.assertEquals(message.tags, DEFAULT_TAGS.get(ERROR))
        self.assertEquals(message.message, USER_ALREADY_EXISTS_MESSAGE)
```

```
# a test to make sure that we do not let users fill out their demographics
twice
 def
test_if_demographics_view_redirects_when_assessment_already_has_demographic
(self):
        create_demographic(self.assessment)
        session = self.client.session
        session['last_question'] = 0
        session.save()
        self.client.session.save()
        kwargs = dict(user_id=self.user.pk,
assessment_id=self.assessment.pk)
        response = self.client.get(reverse('core:demographics',
kwargs=kwargs), follow=True)
        kwargs.update({'number': self.client.session['last_question'] + 1})
        expected_url = reverse('core:question', kwargs=kwargs)
        self.assertRedirects(response, expected_url)
```

3.5 Code Coverage

The coverage.py package was used for code coverage. To set it up, have python installed and run "`pip install coverage`". After this, as we are running django, navigate to the directory with the "`manage.py`" file and run "`coverage run --source='.' manage.py test`". After this, run "`coverage html -i`" to produce an "`htmlcov`" folder which contains information about the code coverage.

We did not test some of the code that was not written by us. Writing unit tests from the django admin interface can be difficult sometimes, so some manual testing was done. Some pieces of code are only run in production mode, but their equivalents were run in testing mode. There is some legacy code that was not tested as it is not currently in use.

Coverage report: 99%

Module	*statements*	*missing*	*excluded*	*coverage* ↓
TheSum/wsgi.py	4	4	0	0%
TheSum/settings.py	71	15	0	79%
manage.py	13	2	0	85%
core/tests/test_models.py	42	4	0	90%
core/tests/test_decorator.py	144	6	0	96%
core/tests/utils.py	123	2	0	98%
core/tests/test_admin.py	497	2	0	99%
TheSum/__init__.py	0	0	0	100%
TheSum/storage_backends.py	14	0	0	100%
TheSum/test_settings.py	2	0	0	100%
TheSum/urls.py	9	0	0	100%
core/__init__.py	0	0	0	100%
core/admin.py	350	0	0	100%
core/apps.py	9	0	0	100%
core/constants.py	33	0	0	100%
core/decorators.py	41	0	0	100%
core/demographic_choices.py	18	0	0	100%
core/forms.py	100	0	0	100%
core/helpers.py	34	0	0	100%
core/middleware.py	52	0	0	100%
core/migrations/0001_initial.py	10	0	0	100%
core/migrations/__init__.py	0	0	0	100%
core/models.py	181	0	0	100%
core/signals.py	65	0	0	100%
core/templatetags/core_extras.py	8	0	0	100%
core/tests/__init__.py	0	0	0	100%
core/tests/test_accessibility.py	136	0	0	100%
core/tests/test_forms.py	57	0	0	100%
core/tests/test_helpers.py	14	0	0	100%
core/tests/test_middleware.py	152	0	0	100%
core/tests/test_signals.py	119	0	0	100%
core/tests/test_views.py	1140	0	0	100%
core/tokens.py	8	0	0	100%
core/urls.py	7	0	0	100%
core/views.py	1106	0	0	100%
visualization/__init__.py	0	0	0	100%
visualization/apps.py	3	0	0	100%
visualization/migrations/__init__.py	0	0	0	100%
visualization/tests.py	268	0	0	100%
visualization/urls.py	4	0	0	100%
visualization/views.py	222	0	0	100%
Total	**5056**	**35**	**0**	**99%**

coverage.py v5.0.3, created at 2020-03-23 15:08

3.6 Installation Instructions

Note: *throughout these instructions,* ***your AWS region might get changed*** *when you go from page to page. If you cannot find your application or environment in the AWS console, it is probably because your region (which you can set in the top right corner of any page next to your user) is not set to the same region as where you made your App. If you change this, then your app should appear.*

Note: *the installation instructions do not cover resetting the database for migration issues, rebuilding the environment and redeploying, ssh, and creating a super user. For this information, see the user manual.*

Note: *you will be charged by AWS for these resources. If you wish to no longer wish to be charged,* ***you must delete the resources and take down your application****. There are instructions at the end of this document for how to delete the S3 bucket, database, environment, and application.*

Before you begin, copy paste the following table somewhere convenient. You will be filling it out throughout the process, and will need it during deployment. Some parts of the table are already filled out. The parts you need to fill out have numbers and letters (such as 17a) associated with them which refer to the steps in the Initial Deployment, Database Info, S3 Bucket, PayPal, and Setting Environment Variable Sections below.

Name	Value
ENV	PRODUCTION
EMAIL_HOST	email-smtp.us-east-1.amazonaws.com
ERROR_REPORT_EMAIL	Any email you want
APP_URL	**This comes from part 17a**
RDS_DB_NAME	**This comes from part 18f**
PRODUCTION_DB_PASSWORD	**This comes from part 13c**
RDS_USER	**This comes from part 13b**
RDS_HOST	**This comes from part 18g**

RDS_PORT	**This comes from part 18h**
SECRET_KEY	uC9OvT@rJ9&sAj&3G980cAp0g$hEe5izR?H59D^VRhCr7r3uTS *Note: you may also pick some arbitrary long piece of text here*
AWS_ACCESS_KEY_ID	**This comes from part 7g or you should already know it**
AWS_SECRET_ACCESS_KEY	**This comes from part 7g or you should already know it**
AWS_STORAGE_BUCKET_NAME	**This comes from part 21a**
AWS_S3_REGION_NAME	**This comes from part 21c**
PAYPAL_IDENTITY_TOKEN	**This comes from part 24**
PAYPAL_RECIEVER_EMAIL	**This comes from part 22**
SEND_EMAIL_PDF	**This comes from part 29**
FROM_EMAIL	info@thepowerofdifference.org
EMAIL_BACKEND_PROD	django_ses.SESBackend
BCC_EMAIL	powerofdifferencetest@gmail.com (test) *Note: this should be whatever email address you want to receive the BCC for the PDF results*

Initial Deployment

1. For the most up to date code, if you have access to it, you should download the code from our GitHub repo! **If you do not have access to the GitHub repo, go to step 2.**
    a. On the master branch page, click “Clone or download” and then click “Download ZIP”.
    b. Unzip this source code and enter your new unzipped folder. From there, you must enter the folder called “src” (if you only see one folder and it is not called “src”, enter that and repeat until you find the “src” folder).
    c. Download the “questions.json” file here: https://drive.google.com/file/d/1rPeHbhr0XBgJQ5pU-W2fuTTfCJH2XAK_/view?usp=sharing

        i. You must be using a valid UVA email for this to work.
        ii. If you are our customer (The Sum), then we have already given you the "questions.json" file. It is not in the GitHub repo as you asked us to keep it private. Please use the "questions.json" file we have previously given you instead of downloading it.
    d. Put the questions.json file into the "src" folder.
    e. **Go to step 3.**
2. Download the source code here: https://drive.google.com/open?id=1ARIBuAHOPS42jwHMzaKHdoOdsq2uthOY
    a. You must be using a valid UVA email for this to work.
    b. Unzip this source code and go into your new unzipped folder.
        i. **Note: you should unzip this into a new folder or else the installation may fail.**
3. Open a terminal (command line) in this folder (the one that you navigated to in step 1 or step 2).
    a. In the terminal, after typing each command, you must press enter.
4. If you do not have git installed, follow this guide here: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
5. If you do not have pip installed, follow this guide here: http://pip.pypa.io/en/stable/installing/
    a. You can check if you have pip installed by running "pip -V". If you get output, it is installed.
6. If you do not have awsebcli installed, follow the instructions here: https://pypi.org/project/awsebcli/
    a. You can check if you have it installed by running "eb --version".
7. Make sure you already have an AWS account and access credentials.
    a. If you are not sure, go to https://aws.amazon.com/ and click on "sign in to the console" in the top right corner.
        i. If needed, make a new account.
    b. Click on your account name and then go to "My Security Credentials".
    c. If asked, say "Continue To Security Credentials".
    d. Go to "Access Keys".
    e. If you already have an access key here and you know both the access key ID and secret key, then you should be able to use those. **In the table that you copied at the beginning, use these to fill out the AWS_ACCESS_KEY_ID, and AWS_SECRET_ACCESS_KEY fields respectively.** Otherwise follow these next few steps.
    f. Click "Create New Access Key".
    g. Next, click "Download Key File".

    i. **Keep this file safe. It is the only way you can get your secret key.**
    ii. **Open the file. You should see an Access Key ID and a secret access key. In the table that you copied at the beginning, use the Access Key ID as the value for AWS_ACCESS_KEY_ID, and the Secret Access Key as the value for AWS_SECRET_ACCESS_KEY.**
8. If you already have an AWS account registered with the awsebcli (what we installed in step 6), then you will have to edit the file located at "~/.aws/config". You can do this with whatever text editor you want. Typically we use vim. To do so with vim:
    a. Have a terminal open.
    b. Type "vim ~/.aws/config" and press enter.
        i. In vim, you can edit the text by typing "i". You get out of edit mode, just press escape. To save the file after exiting edit mode, type ":x" and then press enter.
    c. Edit to add a new profile and number it as the next number that you need.
    d. Edit the file to make it look like the following (actual codes have been turned into XXXX for secrutiy purposes in this guide):
    Here [profile eb-cli] is your pre-existing profile you're using for any other projects and the [profile eb-cli2] is the one you will be using for this project.

    [profile eb-cli]
    aws_access_key_id = XXXXXXXXXXXX
    aws_secret_access_key = XXXXXXXXXXX

    [profile eb-cli2]
    aws_access_key_id = XXXXXXXXXX
    aws_secret_access_key = XXXXXXXXXXX

9. If you had to add a new profile in "~/.aws/config", then all the eb commands will require "--profile eb-cli2" (or whatever you named the new profile linked to the AWS you want to use) to follow them. For example, in step 10, we will use "eb init". If you added profile eb-cli 2, you will need to type "eb init --profile eb-cli2".
10. Go back to the terminal we opened in step 3 and run "eb init".
    a. This will ask you to choose a region. Select a region that you think will be central to the users of your application. We recommend choosing 1 (us-east-1). Type "1" (or the number of the AWS region you choose) and press enter.
    b. You will then be asked for your AWS credentials (type them in when prompted for each part).
11. Work on creating the application:
    a. Type the number for [Create new Application] and press enter.
    b. You will be asked for a name. Give the application a name and press enter.

c. You should be asked if you are using Python. Type “Y” and press enter.
d. You will be asked about which Python version. To use the default (which we recommend), type “1” and press enter.
e. You should be asked to set up SSH. Type “Y” and press enter.
f. You will be asked to make a key pair name. Use the default of “aws-eb” (and just press enter to use this, do not type anything).
    i. If you already have a keypair you want to use or you want to make a new one, then follow the prompts to do so.
g. You will be asked for a key passphrase, choose whatever you want and **remember it**! You may simply just type enter to have no passphrase.

12. Now we want to create our elastic beanstalk (AWS) environment! You will want to use a name that ends in ENV (for readability purposes). In this example, we will use testingENV for our environment.
13. Run “eb create <name from step 12> --database.engine postgres”.
    a. You will be asked for an RDS DB username. Either type a username that you choose and press enter or press enter to use the default which is ebroot . **In the table you copied at the beginning, fill this out as the value for RDS_USER.**
    b. You will then be asked for an RDS DB master password. Type one in and press enter. **In the table you copied at the beginning, fill this out as the value for PRODUCTION_DB_PASSWORD.**
    c. This will take a while to run as AWS must set up our database. Let it run. This may take 20 or more minutes.
    d. As long as the “Create Environment Operation is complete” message shows up, you can disregard errors related to “Environment Variables”. We will fix those later.
14. After this has finished, you should see the “$” prompt again.
15. Open a web browser and go to https://aws.amazon.com/ and login.
16. In “Find Services”, type in “elastic beanstalk”.
    a. Only one option should appear. Click on it!
17. In this guide, we called our application testing and our environment testingENV. Click on the part where you see testingENV. It will likely be red.
    a. On the top bar where you see “All Applications > testing > testingENV”, you should see a URL. **In the table you copied at the beginning, fill this as the value for APP_URL (do not include http://… only use exactly what the URL on this page says).**
    b. In the end, you should be able to find your application at this URL.
    c. On the left hand sidebar, click configuration.

Getting the Database Info

18. Scroll down until you see “Database”.
    a. Look at the endpoint. Remember everything before the first “.” .
    b. Open the link next to “endpoint” in a new tab.
    c. Click on the DB identifier that matches what you remembered in step 18a.
    d. You will need the info on this page to fill out some of the fields in the table you copied at the beginning.
    e. **DB identifier:** fill this out as the value for **RDS_DB_NAME** *(example: aaq62w5zlej09g).*
    f. **Endpoint**: (this is a bit different than the endpoint from the previous pages) fill this out as the value for **RDS_HOST** *(example: aaq62w5zlej09g.cwjhfzdgx2us.us-east-1.rds.amazonaws.com).*
    g. **Port**: fill this out as the value for **RDS_PORT** *(example: 5432).*

Create S3 Bucket for Static Files

19. Now go to https://aws.amazon.com/ and get to the management console like in step 15. If you know you are already logged in, you can go directly to https://console.aws.amazon.com/.
20. In Find Services, type “S3” and click on the option that only says “S3”.
21. Click on “Create Bucket”.
    a. For bucket name, type in a name that gets accepted and that you will use as the name for your S3 bucket. **In the table you copied at the beginning, fill this out as the value for AWS_STORAGE_BUCKET_NAME.**
    b. For region, we recommend using US East (N. Virginia).
    c. Regardless of which region you choose, you need to get the region name as you need it for the **AWS_S3_REGION_NAME** environment variable.
        i. For example, If you choose “US East (N.Virginia)”, then your region name will be “us-east-1” and you will then use “us-east-1” as your environment variable for **AWS_S3_REGION_NAME**.
        ii. Mapping Table:

Region	Name
US East (Ohio)	us-east-2

US East (N. Virginia)	us-east-1
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka-Local)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1

Middle East (Bahrain)	me-south-1
South America (São Paulo)	sa-east-1

d. Keep clicking next and then eventually click “Create Bucket”.
e. You should now see your new bucket listed.

Setting up PayPal

22. Sign up (if needed) and log in to your paypal **business account**: https://www.paypal.com/login. The email used for this paypal account is the **PAYPAL_RECIEVER_EMAIL**.
    a. The needed functionality is only available for business accounts and not for personal accounts.
    b. If you want to use a test business account (as you may not have a real business or you want to do some testing), go here: https://developer.paypal.com/. Click on “Log into Dashboard” in the top right corner. After this, click on “Sign Up" and fill out the form to make your developer account. This will automatically give you a test personal account and a test business account.
        i. If you already have a testing business account, you can use that instead of making a new one.
    c. Once logged in to your developer account, use the left side bar, scroll down to the sandbox section, and click on “Accounts”. On the main page, scroll down to where you see the sandbox accounts.
    d. Next to the “BUSINESS” account, in the “Manage Accounts” column, click on the “…” button.
        i. You should now get to see the email and a system generated password for your test business account.
    e. Use https://www.sandbox.paypal.com/us/signin to log in to your test business account (credentials found above).
23. Use the gear in the upper right corner to navigate to “Account Settings”. Under “Products and Services” select “Website payments.” Next to website preferences press “Update.” Turn “Auto Return” ON, and add the link to the **APP_URL** as the “Return URL”.
    a. For example, in step 17a, you found your **APP_URL** to be abc.com, use abc.com for the “Return URL” in paypal.
        i. If you already have a URL set for the “Return URL”, you do not need to change it.

24. Turn “Payment Data Transfer” ON (Note that “Auto Return” must be turned on before “Payment Data Transfer”). You will need the newly presented “Identity Token” for the **PAYPAL_IDENTITY_TOKEN**.
    a. If you already have “Payment Data Transfer” set to on and your identity token is not present, set it to off and then turn it back on. Your identity token will then appear.

Setting Environment Variables

25. Now we can go back to our old tab from the start of step 18 (you may follow steps 15 through 17 again if you lost this tab).
26. Scroll up in the configuration page (which you should already be on) until you see “Software”. Click on the “Modify” button.
27. Scroll down until you see “Environment Properties”.
28. In “Environment Properties”, one by one, add the Name Value pairs from the table you copied at the beginning.
29. For the **SEND_PDF_EMAIL** environment property, if you want results emails to be sent to users, set this to anything other than OFF. If you DO NOT want results emails to be sent to users, set this to OFF (it must exactly match those three characters, capital, no spaces).
30. Click “Apply” at the bottom of this page. Finally, wait for some time and you should finally see that green checkmark!

Fixing Deployment Issues

31. **IMPORTANT:** In step 30, if you had any issues (such as “WSGI path does not exist”), please do the following:
    a. Navigate back to the Dashboard page for your environment. If you do not remember how to get here, please follow steps 15 through 17.
    b. Under Actions, click on “Restart App Server(s)”.
    c. Click the red box that says “Restart App Server(s)” to confirm this.
    d. This should the WSGI path not found error (AWS has some bugs in it that we must work around). However, now going to your app you will get a 404 error or another web error.
    e. Go back to your terminal that opened in the folder with the code. Type “eb deploy” and press enter. Once this has finished and your app has deployed then you should be done! It may take a few moments to start up, but the URL you got earlier should be accessible and you should find the app running there!

Removing the AWS Resources

32. Deleting the S3 bucket:
    a. From the main page of the AWS console, go to the main S3 bucket page like in steps 19 through 21.
    b. For the bucket you want to delete, click the check box next to it and then press the “Delete” button at the top of the list.
    c. You will see a screen that asks you to type the name of the bucket and then press the confirm button. Note: your environment name will appear on the confirmation page. Follow these instructions.
33. Deleting the database:
    a. Navigate to your database like you did in step 18. If you need to get back to your environment first, follow steps 15 through 17.
    b. Under the actions tab, click “Delete”.
    c. You will see a new page about snapshots and asking you to confirm the deletion.
    d. Uncheck “Create Final Snapshot”.
    e. Check “I acknowledge that…”.
    f. Type “delete me” into the field at the bottom.
    g. Click “Delete”.
34. Deleting the environment (make sure you deleted the database first):
    a. Navigate to the environments dashboard like in steps 15 through 17.
    b. Click on the actions button and then click on “Terminate Environment”.
    c. You will see a confirmation page. To finalize the deletion, type the environment name into the box and press the “Terminate” button. Note: your environment name will appear on the confirmation page.
    d. If you just deleted the database for the environment, it may take a few minutes before the environment gets deleted and disappears from your view.
35. Deleting the application (make sure you deleted the environment first):
    a. Go to the applications page. If you need to, follow steps 15 and 16 to get here.
    b. For the application you want to delete, click on “Actions” and then click on “Delete Applications”.
    c. You will see a confirmation page. To finalize the deletion, type the application name into the box and press the “Delete” button. Note: your application name will appear on the confirmation page.
    d. If you just deleted the environment within the application, it may take a few minutes before the application gets deleted and disappears from your view.

DNS and HTTPS (Optional, but must be done together)

Note: In order to set up HTTPS, we will need to move to a new environment. At the end of this process, you will have to delete your old environment. Make sure to take a snapshot of the database so you do not lose any data!

36. We are assuming you have purchased a domain name from GoDaddy.
37. First, we will make a hosted zone in AWS.
38. From the AWS console main page, search for and go to “Route 53”.
39. In the dashboard on the left, click on “Hosted Zones”.
40. Click on “Create Hosted Zone” and enter the domain name you will be using, a comment (if you want), leave type as public hosted zone, and click “Create”.
41. Go into the hosted zone. You should see an NS record with values. We will need these values in step 52.
42. Go to https://dcc.godaddy.com/domains/ (log in to godaddy.com first).
43. Click on the domain name you wish to use.
44. Scroll down to the bottom of the page and click on “Manage DNS”.
45. Under “Advanced Features”, you want to click on “Export Zone FIle (Unix)”.
    a. In this file, you should see something like the following:
       ; A Records
       @ 600 IN A Parked
    b. Change “Parked” to “1.1.1.1”.
    c. Save the file!
46. In your AWS console, go back to where you saw the records within your hosted zone. Click on “Import Zone File” and upload our file from the previous step.
47. Refresh the page and you should see new records. Click on the A record.
48. Change “Alias” to YES and in the alias target box, enter the URL for the app from step. 17a. Click on the “Save” button to make this change!
49. Go back to the GoDaddy page to “Manage DNS”.
50. Scroll down to the nameservers section, click on “Change”.
51. Click on “I’ll use my own name servers”. This may be towards the bottom as setting nameservers manually (advanced option).
52. Enter the values you got from step 41.
    a. Each value from step 41 should be on its own line and will be its own nameserver here.
    b. You will have to remove the trailing periods from AWS for these to work with GoDaddy.
    c. Note: you may need to press “Add Nameserver” for each nameserver you want to add over 2 nameserver. You will likely need a total of 4 nameservers!
    d. Press Save.

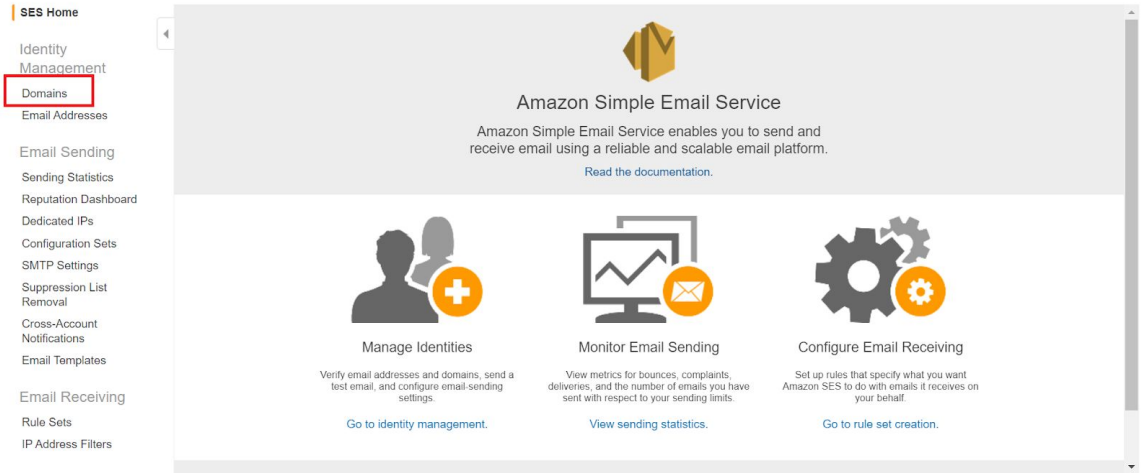
53. Finally, go to the AWS console, go to your environment, click on “Configuration”, and then “Modify” for “Software” (this will take you to where you were in step 26).
54. Change the **APP_URL** to your domain name and save this configuration!
    a. We now have our domain name setup! All we have left to do is setup HTTPS!
55. First, we need to get a certificate for SSL/TLS.
56. Once logged into the AWS console, go to https://console.aws.amazon.com/acm/home?region=us-east-1#/.
57. If you do not already have a certificate for your domain name (which you probably don’t right now), click on “Request a certificate”.
58. Request a public certificate.
59. When adding domain names, add <your domain name> and also add www.<your domain name>.
60. Click “Next”.
61. Choose “DNS validation”.
62. Click “Next”.
63. You can skip adding tags, click on “Review”.
64. Click “Confirm and Request”.
65. Go back to https://console.aws.amazon.com/acm/home?region=us-east-1#/.
66. Click on the drop down for <your domain name>.
67. In the smaller box that says domain, click on the dropdowns for each domain here and click on “Create Record in Route 53”.
68. Your certificate should be verified soon! After it’s verified, follow the rest of the steps.
69. Go to your environment that you wish to move over to HTTPS. Press Actions and select “Save Configuration”.
    a. You will give this configuration a name. Remember it! We will call this <saved-configuration>.
70. Go up to the application level and select “Application Versions” from the bar on the left.
    a. Note the <version-label> for the item that is currently deployed to the environment we are recreating.
71. Go into your terminal. Find the place where you ran “eb init”. If you need help getting here, follow steps from earlier in the guide. You may have to run “eb init” again.
72. Run “eb create <new-environment-name> --elb-type application --cfg <saved-configuration> --version <version-label>”.
    a. <new-environment-name> will be the name for your new environment.
    b. This will run for a while, but it will give you an application load balancer which allows us to use HTTPS.
73. Once the new environment is set up, you will have to go and set up the database environment variables again (by changing the environment, we made a new database). Follow step 18 to get the database info for the new environment. Modify the **RDS_HOST**

and **RDS_DB_NAME** environment variables accordingly in the software configurations for your elastic beanstalk environment. If you forgot how to get to the software configurations, see steps 15 through 17 to get to your environment in a web browser and see steps 25 through 29 to change the environment variables.

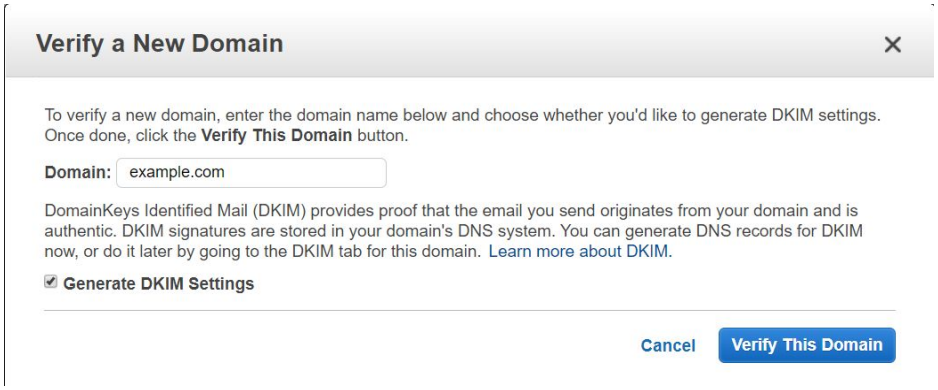
74. Follow this guide: https://medium.com/@j_cunanan05/how-to-redirect-http-to-https-in-amazon-web-services-aws-elastic-beanstalk-67f309734e81
    a. To check which load balancer to edit in the EC2 configurations, you can click on one and then go to the tags tab. It should say the environment name is the name of your new environment.
75. Follow the part of the guide above to delete your old resources for your old environment!
    a. Make sure to make a snapshot of the database so you don't lose any data (if you do not have any data you want to keep, feel free to forgo the snapshot).
76. You may receive the WSGI issues again. If this is the case, restart the app servers like you have done previously. To redeploy, click on "Upload and Deploy", then click on the "Application Version" page.
77. Click the check box for the current app version (this should be the app version we used earlier in our eb create command), click "Actions", and click on "Deploy", click on the environment name you want to deploy to, and then finally click "Deploy".
78. We now have HTTPS!

Configuring the email server using AWS Simple Email Services - for production

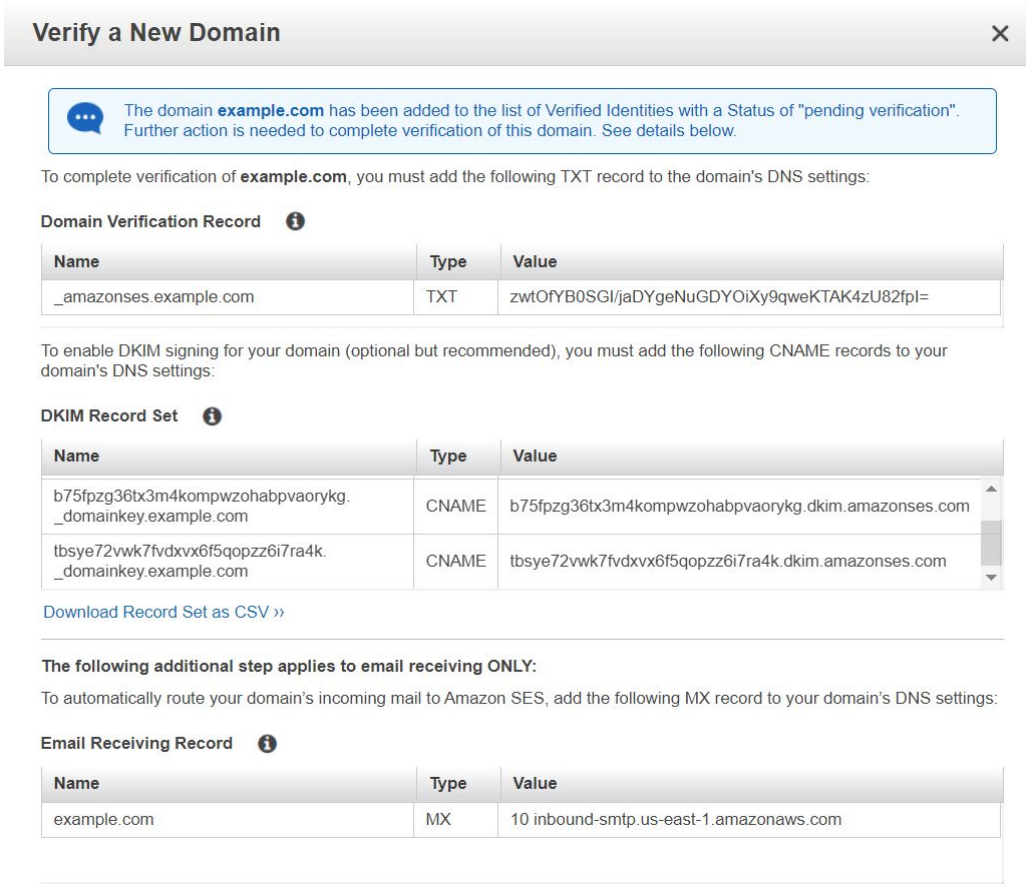
79. We are assuming that Route 53 has been set up.
80. From the main page of the AWS console, search for and go to "Amazon Simple Email Services (SES)".
81. Click on "Domains" under "Identity Management" on the left side of the screen.



82. Select “Verify a New Domain” at the top left of the center screen.
83. In the “Domain” box, enter the domain name. For us, it was “thepowerofdifference.org”.
    a. Also select the checkbox to “Generate DKIM Settings”.



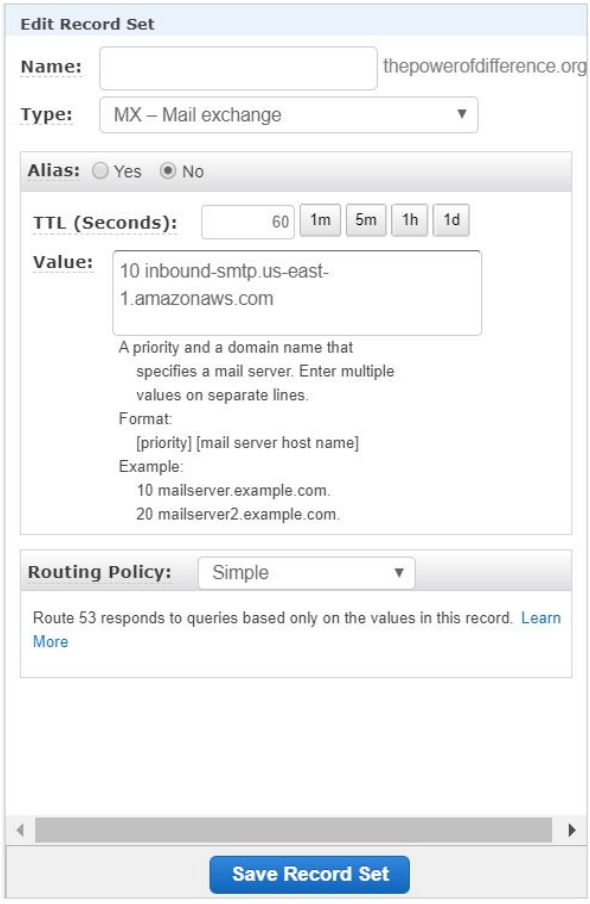
84. The result should look like this after selecting “Verifying This Domain”:



85. These record sets must be updated in the domain’s DNS settings.
    a. If a button to add them automatically using Amazon Route 53 is present, select that.

b. Otherwise, navigate to Amazon Route 53, select “Hosted zones” on the left navigation pane, select the domain name that was picked in the previous section, select “Go to Record Sets,” and create record sets for each of the records in the screenshot above.
    i. Note: The records in the screenshot above will have different names and values. Use the ones that appear on your AWS page.

86. Next, set up an S3 bucket to receive emails for email verification.
    a. Navigate to Amazon S3.
    b. Click on “Buckets” and “Create a Bucket”.
    c. On the create bucket screen, follow the instructions to create a unique name and set the region to the same region as the rest of your stuff. After clicking on the “Block All Public Access” checkbox, click on “Create Bucket” to create your bucket.
    d. Navigate back to Amazon Route 53 to update the record sets once again to add the following record:

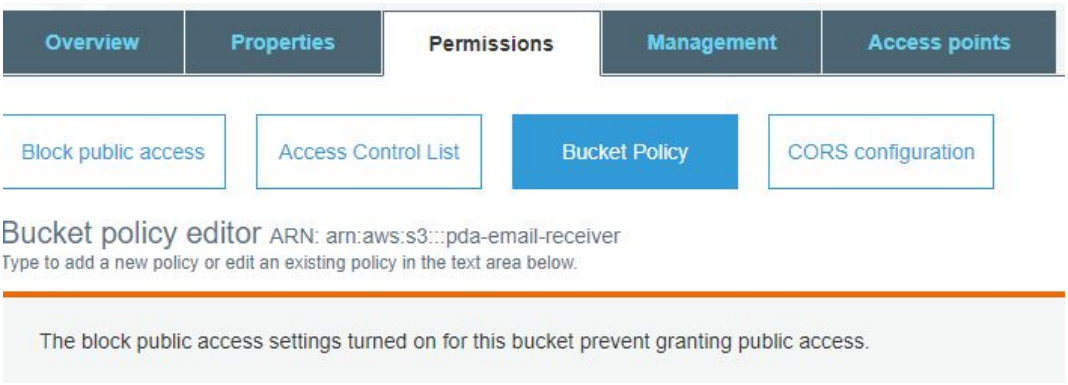


    e. Notes:
        i. Leave “Name” blank. The part after name will match the domain you used when setting this up.

ii. If you wish to use a different region, change the value to use a different region.

f. After finishing up setting the records. Go back to Amazon S3 and click on the bucket you just made.

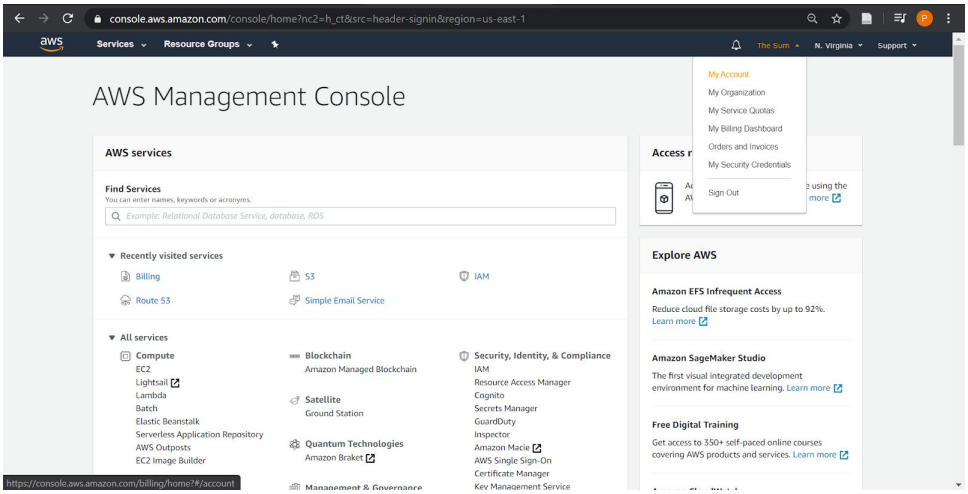
g. Go to “Permissions” and then “Bucket Policy”.



h. Replace “BUCKET-NAME” and “AWSACCOUNTID” and put this in the “Bucket Policy Editor”:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowSESPuts",
         "Effect": "Allow",
         "Principal": {
            "Service": "ses.amazonaws.com"
         },
         "Action": "s3:PutObject",
         "Resource": "arn:aws:s3:::BUCKET-NAME/*",
         "Condition": {
            "StringEquals": {
               "aws:Referer": "AWSACCOUNTID"
            }
         }
      }
   ]
}
```

i. Your AWS Account ID can be found by clicking on “My Account” and looking under “Account” Settings at the top.



ii. Replace “BUCKET-NAME” with the name provided for the S3 bucket.

i. Hit the save button to save the policy.

87. Email Verification:

a. Go back to Amazon SES and click on “Email Addresses” which is under “Domains” in the “Identity Management” section.

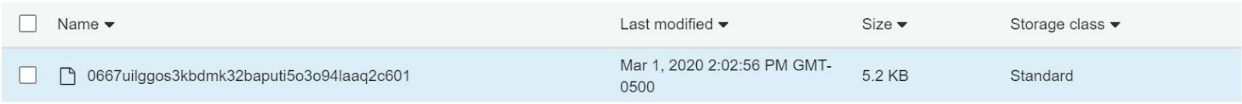
b. Select “Verify a New Email Address” and enter the email address you want to be verified.

i. Note: Use the domain name you verified in the last step (e.g. info@example.com).

c. A verification email will be sent to this domain.

d. Navigate into the S3 bucket by going to AWS S3 and selecting the S3 bucket that was created in the previous step.

i. There should be two emails. One is “AWS_SES_SETUP_NOTIFICATION” and can be ignored. The other will appear to be random numbers and letters and should have a very recent last modified time.

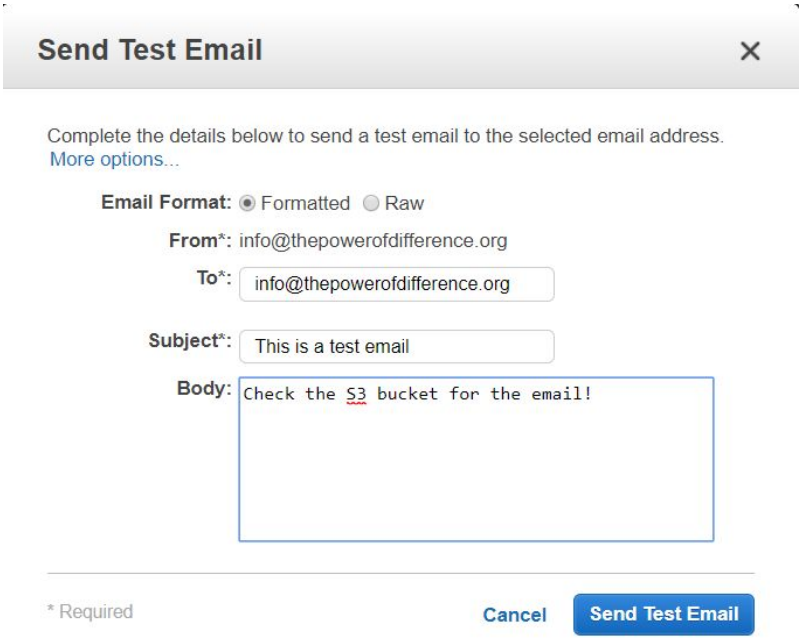


e. Select this email and click “Download.” Open the file that has been downloaded using a text editor, such as Notepad. You may have to find the downloaded file in the downloads folder before right clicking the file and selecting “Open with” to view the contents.

f. Scroll down to the bottom of the file until you find the email contents, which should look something like this:

g. Copy paste the email verification link into a browser to successfully verify the email account.

88. Optional - Send a test message to your domain.
    a. Navigate to Amazon SES, select “Email Addresses”, choose the email address that was verified in the previous step, and click on “Send a Test Email”.
    b. In the “To*:” section of the email, make sure to use your verified domain.

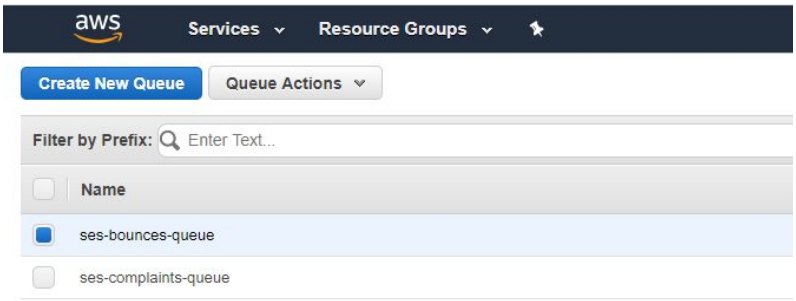


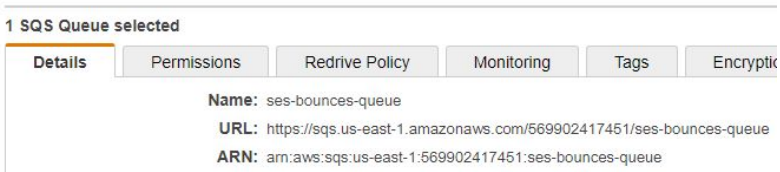
    c. Navigate back to Amazon S3 and view the S3 bucket for receiving emails.
    d. A new email should appear with the contents above.

89. To send emails to unverified emails, a support request must be submitted to get the application out of sandbox mode.

a. Navigate to the AWS Support Center and select on “Create Case” under Open support cases.
b. In the Create Case Info, select “Service limit increase”.
c. Under Limit type, select “SES Sending Limits”.
d. In the contents of the support request, describe the system, what emails are needed for, and how you will comply with AWS SES standards.
e. It may take a while for the support request to be handled. Once handled, the application will be out of sandbox mode and you can send emails to any email address.

90. Handling Bounces and Complaints:
a. Go to Amazon Simple Queue Service (SQS) click “Create New Queue”.
b. Name the new queue something like “ses-bounces-queue”, set it to “standard queue,” and then create the queue.
c. Create another queue with the same parameters called “ses-complaints-queue”.
d. Look at each of the queues and write down their arn values, which can be found at the bottom of the page.





e. Next head to Amazon Simple Notification Service (SNS). Select “Topics” on the left and “Create a New Topic”.

f. Use “ses-bounces-topic” as the name. Leave all other settings as their defaults, then create the topic.
g. Create another topic called “ses-complaints-topic”.
h. Click on the “ses-bounces-topic” and click “Create Subscription”.
i. In this next page set protocol to “SQS”. Then set the end point the ARN value for “ses-bounces-queue” from earlier. Then create the subscription.
j. Do steps h and i again for “ses-complaints-topic”, and set the end point to the ARN value for “ses-complaints-queue” from earlier.
k. After this, go back to Amazon SES and click on domains.
l. Select your domain, and hit “View Details”.
m. Go to the notification tab and hit “Edit Notifications”.
n. Set bounces to “ses-bounces-topic” and complaints to “ses-complaints-topic” and hit “Save Config”.

91. Change the **ENV** environment variable to use AWS email server (production).
    a. Go to the environment variable on AWS. Refer to Step 26 to remember how to reach this page and make sure **ENV** is set to PRODUCTION.

4. Results

The new PDA system has solved many of the problems faced by The Sum. The new system now differentiates between versions of the PDA. It now has a more user friendly interface. The new PDA system now automatically generates and emails PDF results to users. This used to be a manual process that would take 5 or more minutes per PDF, and only one results PDF could be worked on at a time. The PDF generation is now an automated process that takes no more than one minute per PDF. Multiple PDFs can now be generated at the same time. It seems that the old PDA system could not handle many simultaneous users. With AWS, the PDA is scalable. With 6 requests in a 5 minutes period utilizing only 1.3% of the CPU, it has been estimated that the system can handle 200 to 300 simultaneous users, including PDF generations. This is with one instance of the machine. The system has automatic load balancing with a current maximum of 4 machines which should be able to support at least 1,000

simultaneous users. This number can be increased and the CPUs on the machines can be changed in the future to support even more simultaneous users.

The customer uses the system by hosting the PDA for users to take, verifying users, generating PDF results, sending results to users, managing consultants, managing assessment retakes, managing user data, and visualizing user data. Other stakeholders include PDA takers, PDA consultants, and those wishing to perform studies based on PDA results. The PDA takers will use the new system to take the PDA, receive automatically generated results, and schedule a consultant meeting. PDA consultants will use the new system to access the information of users who they work with, instead of the transfer of information being a manual process. Those wishing to study the PDA will be able to use the system in order to have many people take the PDA and receive results in a timely manner. They will have the data easily available to them through the Django admin pages. The Sum, PDA consultants, and those studying the PDA can all use the visualization tools to help understand and study the responses from users.

5. Conclusions

This project demonstrated the importance of wireframes, customer meetings, sprint planning, and sprint review as tools for communication. Clear and timely communication helps move the project along smoothly, while also maintaining a mutual understanding between engineers and clients. In creating this system, the team also learned about software as a tool to benefit society, rather than as a product to make profits. We learned that breakthrough innovations, the kind often exemplified by tech startups, are not needed to bring about meaningful change. Helping those with demographic differences to communicate better is incredibly important. The PDA aims to help remove biases to promote communication and understanding across all people.

While building the new PDA system, the team worked to meet basic accessibility standards. We also worked on basic security to make sure users do not abuse URLs or access data that is not meant for them. This taught the team that software engineering is not always about writing code. Instead, software engineering is about the entire process from design to maintenance. This includes managing how the system responds to users who use the system improperly. This also includes testing code to make sure the system behaves as expected in cases where the system is used both properly and improperly.

Ideally, the new PDA system will support the study of 1000 PDA takers. This will help The Sum to learn more about how people respond to demographic differences. It will also help The Sum start working towards their goal of removing biases. In addition, a study should help validate and promote the PDA.

6. Future Work

Future work includes performing the study of 1,000 PDA takers. It can also include data analysis tools to interpret the data from the PDA besides basic visual graphs. The system is not entirely secure. Enhancing the security of the system and providing stronger checks on users would be beneficial. This could include removing the admin application from the public web or using two-factor authentication for staff member access. Generating results now happens automatically, but users must sit on a web page and wait. Changing this so that users are redirected to the next page while the system generates results in the background would improve the user experience. Since the PDA is still in its early days, there are still many minor changes related to instructions, wordings, etc. that are introduced every few weeks. However, in the current system, a developer must perform these changes. Introducing mechanisms so that the product owner can do it himself would be ideal. Another area of improvement could include

enhancing the scheduling tool, so that whenever a user schedules a consultation, the system records that information in the database, and notifies all related parties of the event. Expanding on the usage of AWS and enhancing the load of work the system can handle would be beneficial.

7. References

Headlee, C. (2015). Celeste Headlee: 10 ways to have a better conversation. https://www.ted.com/talks/celeste_headlee_10_ways_to_have_a_better_conversation/upnext.

MIT License. (n.d.). Retrieved from https://choosealicense.com/licenses/mit/

The Sum. (n.d.). http://thesum.org/.

Treasure, J. (2011). Julian Treasure: 5 ways to listen better. https://www.ted.com/talks/julian_treasure_5_ways_to_listen_better/upnext?language=en

Zenger, J., & Folkman, J. (2016). What Great Listeners Actually Do. Harvard Business Review. https://hbr.org/2016/07/what-great-listeners-actually-do.