Smart Charlottesville: Designing the Future

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

Luke Deni
Spring, 2020.

Technical Project Team Members
Cory Ayers
Luke Deni
Sanjana Hajela
Conner Hutson
Anthony Lancaster
Kajal Sheth
Jared Tufts

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 April 4th, 2020
Luke Deni

Approved _____ Date _____
Dr. Ahmed Ibrahim, Department of Computer Science

Table of Content

Abstract **3**

List of Figures **5**

1. Introduction **6**

1.1 Problem Statement 7

1.2 Contributions 9

2. Related Work **11**

3.1 System Requirements 13

3.2 Wireframes 14

3.3 Sample Code 15

3.4 Sample Tests 20

3.5 Code Coverage 23

3.6 Installation Instructions 23

3.6.1 Creating AWS account to host the system 24

3.6.2 Create an EC2 Instance via Elastic Beanstalk 24

3.6.3 Install and Configure PostgreSQL Server 28

3.6.4 Install and Configure the Application 30

3.6.5 Common Problems with Solutions 30

4. Results **31**

6. Future Work **34**

7. References **36**

Abstract

The technical thesis of Smart Charlottesville carries major significance because it states the methodology used to develop a collaborative application. The Smart Charlottesville application connects U.Va students, U.Va faculty, Charlottesville community partners and Charlottesville residents to communicate the ideas they have for transforming the technology of the city. The application was done to bridge the gap between U.Va, the Charlottesville government, and residents so that there is an open platform to interact with one another and exchange ideas for how to make the quality of life in the city better for everyone. Technological changes towards a smart city include improving transit systems, traffic signals, affordable housing, and many other ideas that the users of this app will be able to share. The question we kept in mind throughout the process of building this application is what is the best way for users at U.Va and across the city of Charlottesville to be able to communicate their ideas and collaborate with other people on their ideas.

We obtained the information needed to develop this application in two ways: by collaborating extensively with our client to understand exactly the types of ideas that were to be shared, and how users would interact with other users. We found that sharing posts and commenting on posts are the most popular ways to communicate thoughts, as many social media platforms are already structured that way. Therefore, we developed the application around a user's ability to make posts on blueprints (ideas) they have for a new project they are researching or working on in Charlottesville. The user can give a name, description, category, attachment, and location for the blueprint, which is saved in a database of projects in that category. We

learned it is easiest to categorize the projects in education, transportation, and many others so users can find a blueprint easily. The accessibility of blueprints matters because there are going to be a lot of users accessing this website who want to learn about the projects in Charlottesville and need an easy way to do so, thus being able to collaborate with others faster. Overall, this application is extremely important to facilitate communication of this city so that changes can be brought from the effortless communication and exchange of ideas.

List of Figures

Figure 1. Home page…….14

Figure 2. User profile page……....14

Figure 3. Projects list page…….....14

Figure 4. Submit blueprint page ……...……14

Figure 5. Expanded project page featuring comments……...15

1. Introduction

When was the last time you saw a problem in your hometown, small or large, that you wished you could get help fixing? This could have been as small as a pothole in your neighborhood, to as big as attempting to clean up the water pollution in your local lake. The implementation of technical online platforms has become increasingly popular for facilitating discussion and collaboration. Online platforms where community members, academics, and local government members can collaborate to solve problems can empower individuals to be catalysts of change and community improvement. Uniting minds and resources across this barrier opens new avenues for innovation and improvement technologically and socially.

This is exactly what our thesis project aims to do: bring together all members of the Charlottesville community to create positive interactions and change for this city. There are a plethora of specialized citizens and resources out there, but most citizens, professors, and students have no idea how to access them. By bringing these different members together in a simple, online platform, we hope to allow them to communicate with one another in a positive way that will allow for change to occur in an easier way. Our goal is to eventually have a system where local officials also use the website to look at the needs of the people, which would allow change within laws and make larger projects even more feasible. Furthermore, as cities are becoming smart cities, this platform will be an excellent way to gain citizen input on what aspects of the Charlottesville community will improve the most from a technological upgrade.

1.1 Problem Statement

Professor Ferguson and Professor Ku are part of the STS department at the University of Virginia, and they are conducting research with their STS 4500 students to develop ideas for transforming Charlottesville into a smart city. These ideas, however, need a platform that can be viewed and contributed to by both academics of the University, community members, and local government officials alike. The research problem to be solved is how to facilitate the efficient communication of these ideas, and others, to the Charlottesville community to improve the city for all. At the beginning of the academic year, Professor Ferguson and Professor Ku utilized Digication, an online portfolio tool for sharing student work online, as the platform to solve the problem. However, it was insufficient for a number of reasons including clunky administration and user access permissions, limitations in size and format of files, poor commenting system, and no cross-linking of material from one portfolio to another, among other shortcomings.

The professors also considered using WordPress, but quickly realized that their database needs and user preferences between different user types were going to be too complex for WordPress to handle. Due to these circumstances, they reached out to our capstone group to help solve these problems by creating a simple, easy to use platform with manageable, customized databases, specific user types, and a way to moderate content at an administrator level to keep the site safe and usable for all Charlottesville citizens.

Our solution is beneficial for a number of reasons. Currently, there is no viable platform that solves this problem of lacking communication and collaboration in Charlottesville, since the research problem demands different user types and custom databases that are unavailable with

platforms such as WordPress. The custom written solution is specifically tailored to address the problem and thus better meets the demands of the problem than existing generic solutions. The public web application provides a safe, non-anonymous site for community members to share ideas of changes they want to see in their community through engaging discussions. Community members can find and connect with resources relevant to their ideas, empowering them to be involved in improvement of the city. In addition, community members are able to bring to light problems in the community that may have otherwise remained hidden by posting their ideas and engaging in discussion.

The website will have a feature for users to submit blueprints for proposed projects where they can also add file attachments such as pictures. These blueprints can be categorized by project type: economy, housing, education, water, or politics. The platform will require users to register and login to submit posts and interact with other users. The users will be able to look at the projects and comment on them, mark projects as "favorites" for easy access later, and connect with the authors of the blueprints via email. Community members can also submit smaller problems around the city to gain attention from other members so they can be fixed. There will be an "about us" tab where interested visitors can get in contact with the creators of the site and learn more about this initiative. The landing page will have a map that shows the Charlottesville area with ongoing projects pinned so users can explore projects in different areas by clicking specific pins on the map. Finally, there will be a resources tab describing places users can go to learn more about projects and current city work in general.

1.2 Contributions

We made great contributions to the solution of the problem statement. We were able to create a web based application that allows academics and community members to collaborate in the sharing of ideas for improvement of the city. The platform is much more conducive to letting different types of users not only share proposed projects with one another, it also lets them communicate through commenting on the website as well as emailing one another. We also have made the website much more visual, even from the home page you can view the exact locations on Google Maps of the current proposed projects to see what people are already working on, including what is happening near where you live. Along with this, our contributions also include a resources tab to try to provide as many resources as possible to users of the site to help with what they are working on. This includes helpful tools that range from U.Va resources, to information about the community, water management, and infrastructure.

One of the other main goals of our project was to provide an extremely positive site on which our users can interact. We have done this in a few different ways. First, all content must go through an administrator's approval before it is posted to the official site where normal users can see. This was done to ensure that inappropriate posts or those hurtful towards others were not being made on the website. Along with this we implemented a way for administrators to pre-approve trusted users, such as government officials or others associated with the Smart Charlottesville project, so that administrators would have a reduced load to approve and would not have to worry about trusted sources posting inappropriate things. This is done by making certain email addresses not require approval before those users sign up. We also made sure to not include any dislike buttons on the platform in an attempt to keep all interactions as positive as

possible. This coincides with the favorite button, where users can add different projects to their "favorites" to be viewed easier later on.

Finally, along with being able to share and publish projects, we also accomplished the goal of providing community members with the ability to submit problems that they see around the city. This is a way for their problems and concerns to be heard in hopes of having them solved quicker. This could be as small as a pothole in their neighborhood to a larger community issue that will take other members' help to solve. All of this was planned so we were not only helping make Charlottesville better with new projects, but also fixing up parts of Charlottesville that are in need of some help. All of these contributions we have made to our project have created a collaborative, interactive site that we are proud of.

2. Related Work

There exist platforms and initiatives currently whose missions are to achieve a similar goal. OpenGrounds seeks to "connect the University, the Charlottesville community, and global partners to develop the knowledge, tools, and behaviors that will shape the future" (University of Virginia, 2019). Without an online collaborative environment, however, OpenGrounds fails to take advantage of the internet as a network to easily engage community members across the region and instead relies primarily on a physical studio space for interaction. The City of Harrisburg's Comprehensive Plan process, beHBG, is a "transparent and inclusive community initiative" that "gives the public an opportunity to forge a roadmap" for the development of the city (beHBG, 2016). This initiative is a close analog to the solution we are developing as it allows community members to directly collaborate and post ideas about open spaces, mobility, housing, resources, and more. However, no such platform exists for the Charlottesville area. There are generic solutions to website creation such as WordPress and Wix, however these services are inadequate for the aforementioned problem statement, as different user types and custom databases are needed and are not available with these platforms.

3. System Design

The high level goals of our system are to allow community members and students to bring to light ideas for community improvement, and to facilitate collaboration between communities and academics in the investigation and manifestation of such ideas. This system allows for the potential pooling of resources between the University and community that has previously been difficult due to communication barriers. Community members should be able to propose projects and challenges that need solving with related information. Students should be able to propose projects that they think will benefit the community, as well as view community posted challenges to which they may be able to lend their technical expertise. Community members and students alike are able to then have a dialogue about both proposed challenges and proposed projects to create a solution that will truly be to the benefit of the city. The system seeks to be a place where community members go to voice their ideas and challenges with an intent to solve them in ways that were not previously accessible with the given resources. Administrators should have full control over the application. This includes being able to moderate posts, hide posts, delete posts, change user permission levels, create user accounts, and mark certain users as requiring or not requiring moderation. Administrators can also act as normal users, by creating posts and favoriting them. They can also define user sign-up and preapproval rules, such that users with a specific admin-submitted email or email suffix are automatically assigned a desired user role (admin, student, or community member) with the matching permissions upon sign up.

The system is under the MIT license, which has very few restrictions. It allows for the code we produced in Smart Charlottesville to be used in any way such as "...use, copy, modify,

merge…" (OpenSource). The main language we will use to develop the system is Python, utilizing the Python-based Django Web Framework. For the design of front-end web pages we will utilize HTML and CSS, and employ PostgreSQL for the implementation of the database back-end because of its easy interaction with Django. We chose Python and the Django framework because of its ease of use, complimented with great online documentation, while also being an extremely powerful framework for building full stack web applications.

3.1 System Requirements

Gathering system requirements is imperative to best understand the goals of the client and to facilitate the work of the developers to meet the needs of the client. Listed below are the capstone group's minimum, desired, and optional requirements:

Minimum Requirements:

1. As a user, I want to be able to comment on a blueprint to give my support or feedback.
2. As a user, I want to be able to filter through blueprints based on what category they fall under.
3. As an administrator, I should be able to manage blueprint content by hiding or removing it.
4. As an administrator, I should be able to manage the privileges of other users (students, community partners, and community members).
5. As a student, I should be able to create my own blueprint space so that others may view it.
6. As a student, I should be able to view other students' blueprints.
7. As a community member, I should be able to leave comments on a students blueprint.
8. As a community member, I need to be able to post blueprints.
9. As a community member, I need to be able to like specific comments or blueprints.

Desired Requirements:

1. As a user, I should be able to search for keywords that define the type of blueprints posting I want to look at.
2. As a user, I should be able to view blueprints based on specific location

Optional Requirements:

1. As a user, I should be able to comment on other comments.
2. As a student, I should be able to tag my post with specific categories.

3.2 Wireframes

Wireframes are essential in initial system design and refinement in gathering client feedback. Below are the wireframes we developed for the system.

Figure 1
Home page

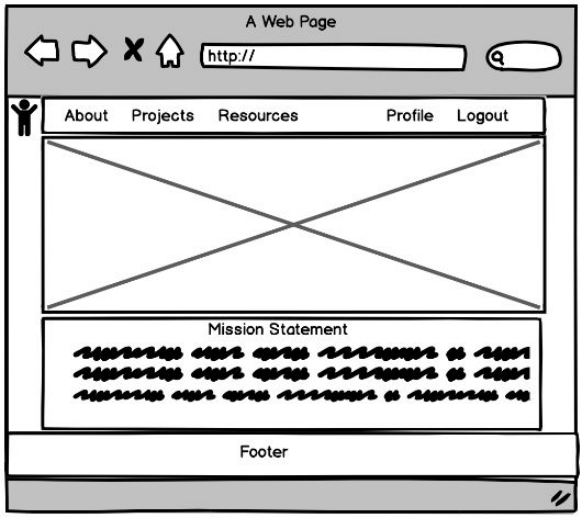


Figure 2
User profile page

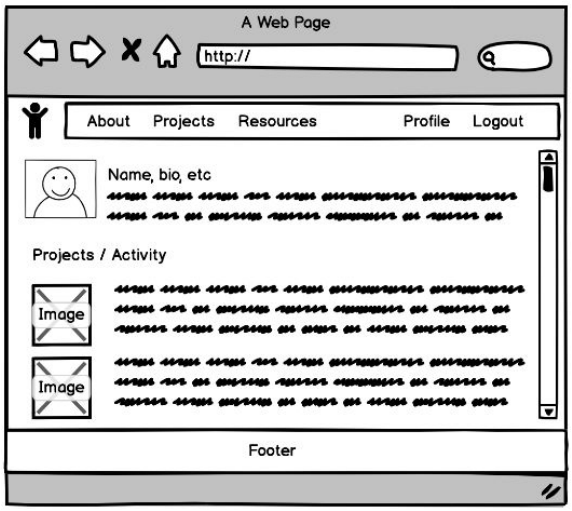
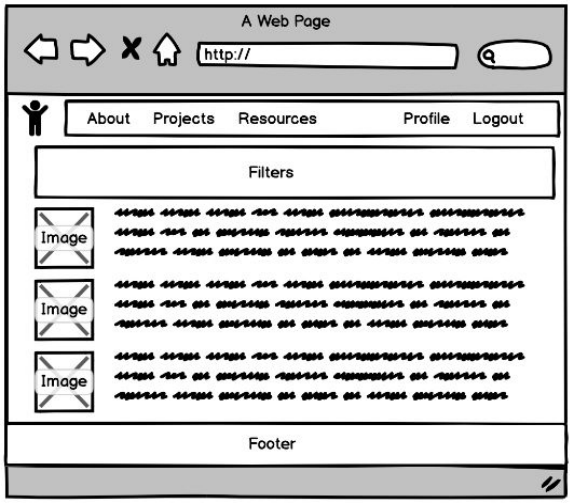


Figure 3

Figure 4
Submit blueprint page

Projects list page



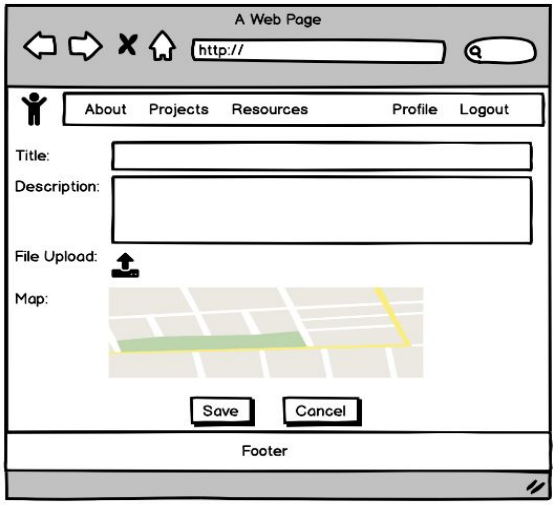
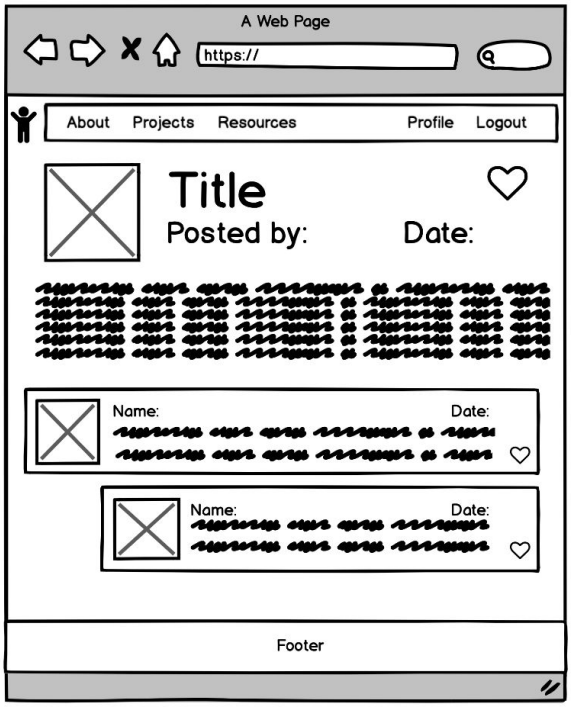


Figure 5

Expanded project page featuring comments



3.3 Sample Code

Models

Below is the code for the Post model, which defines the fields that characterize a post that the users will submit. It also, notably, links the posting user to the created Post. These fields were created based on the feedback for requirements received from our clients.

```
class Post(models.Model):
    title = models.CharField(max_length=100)
    background_literature = models.CharField(max_length=30000, default='')
    research_questions = models.CharField(max_length=10000, default='')
#description of the stakeholders
    stake_holder_desc = models.CharField(max_length=30000, default='')
#A file for the stakeholders
    stake_holder = models.FileField(null=True, blank=True)
    value_mapping = models.FileField(null=True, blank=True)
    bibliography = models.CharField(max_length=20000, default='')
    Vision_Statement = models.CharField(max_length=20000, default='')
    author = models.ForeignKey(
                  User,
                  on_delete=models.CASCADE,
                  default=None,
                  null=True,
      )
    comments = models.ForeignKey(
                  'Comment',
                  on_delete=models.CASCADE,
                  related_name="post_comments",
                  null=True
      )

    def __str__(self):
        return self.title
```

Below is the code for the Comment models, which users submit on a Post. They can be submitted on a full project or on a smaller problem in the community, which is why the class contains both a post and problem field. Other things are also included, such as the author of the comment, the date it was created, and the number of likes the comment has.

```
# model for comment on post
class Comment(models.Model):

    post = models.ForeignKey(
            Post, on_delete=models.CASCADE,
```

```
            related_name='comment_comments',
            null=True)   # post on which this is a comment

    problem = models.ForeignKey(Problem,
                  on_delete=models.CASCADE,
                  related_name='comment_comments',
                  null=True)   # post on which this is a comment
# author of comment
    author = models.ForeignKey(User,on_delete=models.CASCADE)

 # body of comment
    text = models.TextField(max_length = 1000, default = '')

# track time created
    created_date = models.DateTimeField(default=timezone.now)

# allow for comment approval
    approved_comment = models.BooleanField(default=False)

    num_likes = models.IntegerField(default=0)

    def approve(self):
        self.approved_comment = True
        self.save()

    def __str__(self):
        return self.text
```

Below is the code for the Problem model. Different from posting projects, problems allow the user to submit details about smaller changes they want to see in certain areas. Listed below are also the properties, with one of them being the ability to submit a problem request anonymously.

```
# model for problem post
class Problem(models.Model):
    short_description = models.CharField(max_length=100, default='')  #
Short_description = Title of idea
    detailed_description = models.CharField(max_length=1000, default='')  #
Detailed_description = Summary of idea
    possible_location = models.CharField(max_length=100, default='')  #
Possible_location = location in cville of idea (if appropriate)
    expertise_requested = models.CharField(max_length=1000, default='')  #
Expertise_require = what UVa resources do they need / want?
    your_name = models.CharField(max_length=100, default='') # your_name =
name of poster
    your_contact_info = models.CharField(max_length=100, default='') #
your_contact_info = contact info for poster
```

```
    hidden = models.BooleanField(default=False) # true if a post is hidden by
an admin
    flag = models.BooleanField(default=False) # true if a post has been
flagged for being anonymous
    author = models.ForeignKey(
                User,
                on_delete=models.CASCADE,
                default=None,
                null=True,
      )
```

Views

The view below is for creating a Post. The first line that is written shows that for a post to be created, and for this function to even be accessed, the user must be logged in, as only a logged in user can make a post since posts cannot be made anonymously . After that, a Django form gets created which refers to a template that was made in the forms.py file. If the form does get created, which means all the data was valid, then all of the information that was retrieved from the front end gets cleaned (placed in the correct format) and appropriately saved into a Post object and at the end saved into the database as a Model.

```
@login_required
def create(request):
      form = CreateForm(request.POST or None, request.FILES or None)

      if form.is_valid():
            obj= Post()
            obj.title= form.cleaned_data['title']
            obj.category= form.cleaned_data['category']
            obj.background_literature=
form.cleaned_data['background_information']
            obj.research_questions=
form.cleaned_data['proposed_questions_and_resource_needs']
            obj.stake_holder_desc =
form.cleaned_data['stakeholder_and_community_description']
            obj.bibliography = form.cleaned_data['inspiration_and_references']
            obj.Vision_Statement =
form.cleaned_data['Project_or_Vision_Statement']
            # latitude and longitude are used to plot the location on the map
            obj.latitude = request.POST["latitude"]
            obj.longitude= request.POST["longitude"]
            obj.author = request.user
            obj.save()

            return redirect('/projects')
```

```
      context = {
            'form': form,
    }
      return render(request,'create.html', context)
```

The view below is to add a project to your favorites. Again, this can only be done if a user is logged in, because this information needs to get saved. For a project to be saved to a particular user, it needs to first be retrieved by its unique ID.

```
@login_required
def addFavorite(request):
      try:
            post = Post.objects.get(pk=request.POST['post'])
      except:
            return redirect('/projects')

      current_user = request.user
      current_profile = current_user.profile

      current_profile.favorites.add(post)

      next = request.POST.get('next', '/')
      if len(next) > 1:
            return HttpResponseRedirect(next)
      else:
            return redirect('/projects')
```

Forms

Below is the form for creating a project. This is a Django Form, and these are useful because along with being built into Django, it allows for easier testing by allowing the developer to call properties that regular HTML forms might not be able to call. Django forms are also useful because they store all the information straight to a model. For example in this form, the information that gets collected from the front end gets saved as unique fields for the Post model. This form is then saved to the database in a function in the views.py file.

```
class CreateForm(forms.ModelForm):
    title = forms.CharField(widget=forms.TextInput(
        attrs={
            'class': 'form-control'
```

```
        }
    ), required=True)

    CATEGORIES = (
    ('Economy', 'economy'),
    ('Education', 'education'),
    ('Housing', 'housing'),
    ('Water', 'water'),
    ('Politics', 'politics'),
)
    category = forms.ChoiceField(choices=CATEGORIES, required=True )

    Project_or_Vision_Statement = forms.CharField(widget=forms.Textarea(
        attrs={
            'class': 'form-control'
        }
    ))
    background_information = forms.CharField(widget=forms.Textarea(
        attrs={
            'class': 'form-control'
        }
    ))
    proposed_questions_and_resource_needs =
forms.CharField(widget=forms.Textarea(
        attrs={
            'class': 'form-control',

        }
    ))
    stakeholder_and_community_description =
forms.CharField(widget=forms.Textarea(
        attrs={
            'class': 'form-control'
        }
    ))
    inspiration_and_references = forms.CharField(widget=forms.Textarea(
        attrs={
            'class': 'form-control',

        }
    ))
    # location= forms.CharField(widget= forms.TextInput(
    #     attrs={
    #         'class': 'form-control',
    #     }
    # ))
    class Meta:
        model = Post
        fields = ('title', 'category', 'Project_or_Vision_Statement',
'background_information', 'proposed_questions_and_resource_needs',
'stakeholder_and_community_description', 'stake_holder',
'inspiration_and_references', )
```

3.4 Sample Tests

Testing is very important to figure out the validity of your code. Testing helps find edge cases or points in your code that might be breaking by going through all possible scenarios in which the code could be run. We did our testing through Django since our application was built on the Django framework as well so we put all of our tests in the test.py files. We had Travis connected to our GitHub repository for continuous integration support, so every time we pushed any code, it ran through our test files and confirmed if our tests were all passing or invalid.

Tests must be written for each function of the code in order to ensure that the program is working completely. One variety of tests that we utilized was for testing various post models. These tests would create different objects and ensure that the fields in the database were set correctly. This is an example of this type of test for a single project:

```
      def setUp(self):
                  Post.objects.create(title="Single Project 1",
                  Vision_Statement="This is the first project",
                  research_questions="How do we make it out?", pk=1)

      def test_vision_statement2(self):
            my_project = Post.objects.get(pk=1)
            self.assertEqual(my_project.Vision_Statement, 'This is the first
project')
```

Another set of tests included testing various parts of the log in system. These tests involve creating users and operating different log in functions to ensure they work properly. This example tests shows that if a username and password combination map to a particular user, then the system will log in properly:

```
      def setUp(self):
            user = User.objects.create(username='testuser')
            user.set_password('testing321')
```

```
            user.save()

      def test_login_valid(self):
            c = Client()
            logged_in = c.login(username='testuser', password='testing321')
            self.assertTrue(logged_in)
```

Tests were also used to show that the like system in place for comments and posts was properly functioning. These tests create a user and an object to be 'liked'. From here, the object is liked by the user and the test checks to ensure that the linking occured correctly. This example tests if a user can correctly like a comment:

```
      def test_like_owner(self):
            user = User.objects.create(username='testuser', pk=1)
            post = Post.objects.create(title="Single Project 1",
                  Vision_Statement="Vision Statement",
                  research_questions="Research Questions", pk=1)
            comment =
Comment.objects.create(post=post,author=user,text='comment text',
created_date=self.__class__.comment_time, approved_comment=True)
            like = Like.objects.create(user=user, comments=comment)
            self.assertEqual(like.user.username, user.username)
```

The application deals with various levels of user permissions. It was important to test approvals for the various levels and ensure that users could register properly for their accounts. This example shows that a person registering with a virginia.edu account will be able to create a student account and automatically be able to post without moderation:

```
      def register_student(self):
            c = Client()
            logged_in = c.login(username='admin', password='admin')
            self.assertTrue(logged_in)

            preapproval_data = {
            'email' : 'test@virginia.edu',
            'email_qualifier': 'ends with',
            'user_type': 'Student',
            'requires_moderation': 'False'
            }

            c.post(reverse('preapproveUsers'), preapproval_data)
```

```
register_data = {
'username' : 'student',
'password1': 'testing321',
'password2': 'testing321',
'email': 'test@virginia.edu'
}
# signing up with a virginia.edu email should now
# automatically be set to student user type as specified above
self.client.post(reverse('register'), register_data)

user = User.objects.get(username="student")
self.assertEquals(user.profile.user_type, "Student")
self.assertEquals(user.profile.requires_moderation, False)
```

3.5 Code Coverage

To measure code coverage, the command "`pip install coverage`" was used. Once in terminal, open up the files till the application is opened at the level of the manage.py file. Then run the command coverage run --source='.' manage.py test **application name.** Within our project we created two Django applications so we had to run the command "*coverage run --source='.' manage.py test application users*", with application and users being the names of our two Django applications. After this first command is run, the coverage report can be examined using the command "*coverage report*". After running this command, there is a layout of statistics for each file in the application and the report gives statistics on each of these files. Finally, the command "*coverage html*" generates an html file in the directory so that the coverage report can be opened up on a browser and gets saved in the htmlcov directory under the index.html file. Along with being able to open this file up on a browser, the html file allows the user to click on the file names so they can see exactly what parts of the file are and are not being covered by tests. Based on whatever parts of the files aren't being covered, it helps the developers know

what new tests to make. Our code coverage is up to 99%, and the last 1% of code that is not tested is within Django files such as “wsgi.py” that are not testable.

3.6 Installation Instructions

This subsection lists the installation instruction guide for the Smart Charlottesville Web Application. This guide will be split into the following parts for simplification:

1. Creating AWS account to host the system
2. Create an EC2 instance via Elastic Beanstalk
3. Install and configure PostgreSQL server
4. Install and configure the application
5. Common problems with solutions

You can be using Ubuntu, Windows, or Mac OS for this guide, though the commands might slightly vary if using Windows.

3.6.1 Creating AWS account to host the system

To register an account on AWS, go to their website at https://aws.amazon.com/ and click on “Create a Free Account”. Follow the instructions on the website as they guide you through creating the account. Choose a personal account over a professional. You will have to enter a credit card during the process and AWS will charge your card one dollar. Choose the ‘basic plan’ option, and then sign into the console using the login information you chose.

3.6.2 Create an EC2 Instance via Elastic Beanstalk

Retrieve our application from

https://drive.google.com/file/d/1aTW458zvYV8aV1cTxSZDYa-HEISL7Kqr/view?usp=sharing .

Unzip the file and use the terminal to run the following command:

cd src/smart_cville

This is our django project.

Then, create the initialization for the beanstalk environment. To do this, run the command:

mkdir .ebextensions

This will create a directory that AWS will look for to see the configurations.

Run the command:

cd .ebextensions

Then create a file called django.config, which will take the following option_settings:

aws:elasticbeanstalk:container:python:

WSGIPath: smart_cville/wsgi.py

You will need to install the EB CLI. The best way to do this differs by operating system.

Windows:

Install the EB CLI by running the command:

git clone https://github.com/aws/aws-elastic-beanstalk-cli-setup.git

Once this repository has been cloned, switch into the directory and run the installer via

.\aws-elastic-beanstalk-cli-setup\scripts\bundled_installer

Mac:

Install the EB CLI via the command:

brew install awsebcli

Redirect into the .ebextensions folder, if not already there. Create the file packages.config.

In order to use psycopg2, create packages.config within the .ebextensions folder. In the file, include:

packages:

yum:

postgresqlxx-devel: []

Replace xx with the version of PostgreSQL you want to use. For example, using 9.4 would be 94. This is what we used in our deployment.

In the .ebextensions folder, create the file python.config. Put in the file:

container_commands:

01_migrate:

command: "source /opt/python/run/venv/bin/activate && python manage.py migrate --noinput"

leader_only: true

Assure that all files in the .ebextension have the correct tabs, and are not spaces. This is very important for the python.config, django.config, and packages.config.

If in the .ebextensions folder, run the command:

cd ..

This will bring you back into the smart_cville directory. Else, route yourself to the smart_cville directory.

Initialize the elastic beanstalk via the command:

eb init -p python-3.6 django-tutorial

If prompted to answer questions, answer them (using mostly default and choosing Y to create ssh access). If not prompted to answer questions, run the command:

eb init -i

Which will prompt the same questions above.

If this is your first time using the '**eb**' command, it will require you to enter user information for your aws account. To do this, go to your account dropdown (where your username is) in the top right of the AWS console and press My Security Credentials. The information is under "Access keys (access key ID and secret access key)", and you can create a new access key and press download key. Open the downloaded file and it will have your access key id and secret key.

Then, create the environment via the command:

eb create

This will prompt you to insert information about the beanstalk environment you wish to create. Choose the default features for each selection. Choose no for Once completed, information about the beanstalk environment will be outputted and if you want to make fleet instances. Make note of the region in which it was created (west-1, west-2, east-1, east-2), eb init defaults to west-2.

Once completed, run the command:

eb status

Then locate the CNAME, which is the URL which was created for the instance. Take this url and go to the smart_cville/smart_cville/settings.py file in the django project and add it to ALLOWED_HOSTS=[].

Errors here likely include syntax on the configuration files in the .ebextension folder, so make sure you are using the correct tabs and spacings. Events will keep printing if successful, until the elastic beanstalk instance is created.

Once this is added, save the django project and run:

eb deploy

This should have successfully deployed the applications. Running **eb open** will open the application browser but there will be errors due to the database not being set up.

3.6.3 Install and Configure PostgreSQL Server

Go on the AWS console and make sure you are in the correct region (that was made note of previously). You can change your region to the correct one in the top right of the console.

Search for ‘Elastic Beanstalk’ and press on the instance you created. Go to the configurations tab, scroll down to ‘database’, and click ‘Modify.’ Choose a postgres engine, and configure the username and password and leave all others as the default field. Then, press submit and wait for the database to be created. Once it is, press on the RDS instance and go into the file smart_cville/smart_cville/settings.py in the Django project. Update the name (with RDS name), host (with the endpoint), and username / password with the ones you inserted above.

Django projects require migrations to be migrated. To do this, go into the .ebextensions directory and create python.config. Insert the following yaml:

container_commands:

    01_migrate:

command: "source /opt/python/run/venv/bin/activate && python manage.py migrate --noinput" leader_only: true

Redeploy the updated django project via the command (make sure in the smart_cville directory):

eb deploy

Run the command:

eb open

This will allow you to see the updated project.

3.6.4 Install and Configure the Application

Once the database is set up, you should now have a fully functioning application on the ec2 instance. A user account can be registered and logged in, and new posts can be formed, etc.. All the functionality of the pages should now work.

3.6.5 Common Problems with Solutions

If installing the EB CLI on Windows, and you get an error stating something similar to:

Could not install packages due to an EnvironmentError:

Make sure the command window is running in administrator mode. To fix other common errors, go to https://github.com/aws/aws-elastic-beanstalk-cli-setup.

If it successfully deployed, but your website is not showing on the site, run the command:

eb logs

This will show the logs, and thus you can track the error messages.

4. Results

This system is able to solve all of the problems stated by the stakeholders. The previous solution did not have tiered users, where the new solution contains an Admin, Student, and Community Member level. Each one of these levels helps to organize the different submissions and determines the permission level when it comes to posting projects. The website is also accessible by the wider Charlottesville community, and they can post community challenges and full projects that they think will be beneficial to them. This type of system did not exist with the previous solution being used by Professors Ferguson and Ku.

Professor Ferguson and Professor Ku will use the system in two ways. First, they will use it as a way to teach their students to think about the problems in their community by allowing them to post improvements to current systems they observe in Charlottesville. Second, they will use the system to promote a connection between the resources at U.Va and the greater Charlottesville community. Currently there is nothing to bridge this gap, so Professor Ferguson and Professor Ku plan to use this as a platform for communication on how to best use the money and intellectual resources at U.Va's disposal.

The students in Professors Ferguson and Ku's sections of STS 4500 will use this platform to post their solutions or improvements to different problems they observed in the greater Charlottesville area. Through this website they will be able to get immediate feedback from other students, professors, and community members. On the previous website, Digication, they could only really get feedback from their professors and the other students. While this had its benefits, the ability to get feedback from actual members of the community that these solutions would

directly affect is invaluable. The Charlottesville community will also benefit from this, since they will be, along with the students, able to post different projects they think U.Va could work on that would be beneficial to them. Overall, the Smart Charlottesville application is a great improvement for all the stakeholders and provides a method of communication between U.Va and the community that was not there before.

Through the current system of using digication, viewers are limited to Professor Ku and Ferguson, as well as other classmates. This limits the outreach of student work to around 30-100 people. Through this new platform, student work has the chance to reach the whole Charlottesville community. This includes the student body and faculty (roughly 40,000 people) and the city (roughly 50,000 people). Although it is not guaranteed the whole community will use it, even getting 5% engagement would be a drastic improvement. Thus, the impact that Smart Charlottesville can have on not only making student work more visible to the community, but also increasing communication between stakeholders, is very large.

5. Conclusions

The most profound impact we want for any reader to realize is that we built a website that will be an important stepping stone for addressing a larger scale problem. It is impossible to completely dissolve a problem. Changes can not be done in just one day. Large, fundamental changes are a process that require small, progressive steps and in this way our website will be great for the improvement of the Charlottesville community. We constructed a website that is the stepping stone for our clients and the community to use for greater change for the city. The system works to bridge the divides between community members, academics, resources, and others so that collaborative efforts can be made to address community challenges. It is also important to note that the system, though tailored to address the needs of our client and the community, is easily portable to other geographic locations or contexts that would benefit from improved collaboration and communication between groups in solving a breadth of challenges.

Smart Charlottesville is about making Charlottesville a better place, but also how to strengthen communities through technology. This idea applies to communities all around the world. Technology today is used to bring together people from all across the globe, but less frequently to bring together community members. Through Smart Charlottesville, we are able to show steps in revitalizing communities by using technology.

6. Future Work

For future work on this project, the most important plan of action is to get this to the community. We have worked to make it fully functional, giving admin access to our clients and constructing both a frontend and backend. Now, the biggest thing is for the website to go live so that it can have the profound effects it was built for. Only when the website is well marketed to clients will legislators get full feedback on what steps to take for the betterment of the community.

Further research should be done to determine the best method when gathering feedback from communities and the best way to filter out the best suggestions and ideas. This is because when blueprints for projects get posted, there can be strong disagreements between community members, so having a plan in place that allows for the evaluation of multiple ways of tackling a problem is essential. Also, with many blueprints being posted, knowing how to prioritize which ones are the most important is key. More research is definitely needed for this process.

In addition to researching how to best observe community feedback, it is essential for the platform to monitor who is using the application and how often. In order for the website to be as effective as possible, it is imperative to attract the largest community base that it can. Research can and should be done to see how similar systems were marketed in other cities as well as what would be the most effective and incentivizing way to communicate the system to the intended participants. It is only when the community is fully engaged that the platform can begin to meet the goals that it was created to fulfill. This could involve a multi-step process, marketing it to students through a U.Va campaign, to the government through city council meetings, and to the community through posting it around the community. Our hope is for this product to reach as

many citizens of Charlottesville as possible to help make this city even better than it already is. Ideally, with this platform, Charlottesville can look forward to a bright future where the citizens have a say in guiding the direction of the city, and U.Va has the ability to focus its efforts to help Charlottesville in the most effective way possible.

7. References

beHBG (2016). About. *City of Harrisburg*. http://behbg.com/about

Opensource (2020). *The MIT License.* Retrieved from opensource.org

University of Virginia (2019). OpenGrounds Mission. *Rector and Visitors of the University of Virginia.* https://opengrounds.virginia.edu/mission