Medella

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

Jackson Kennedy
Spring, 2020.

Technical Project Team Members
Audrey Fifer
James Hamil
Grace Huang
David Mehani
Aditi Takle
Bernice Wu

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 _____
        Jackson Kennedy

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

Table of Content

Abstract

Medella is a corporate wellness application for distributing wellness related content to employees of all enrolled businesses. Companies are able to sign-up to use Medella, which will give their employees the ability to create accounts with access to all the content that the Medella business team has curated for them. Various mediums of content will be used including videos, quizzes, newsletters and blogs. The scalability of this system is much greater than the client's current system as it removes the need to maintain a manually kept list of 30 or so companies and what is being provided to each. Another aspect of Medella's scalability is its ability to track information about engagement with all content. This tracking is done for both the type of media as well as the tags associated with a piece of content. Thus, Medella business team members are able to see what content users interact with the most in order to fine tune what content they provide in the future. Companies that have a history of engaging with certain mediums and tags can be provided more of that content. On top of this, if a user interacts with a certain tag, they have all content sent to that tag put into their personal archive in the future, regardless of whether or not it is sent to their business. Personalization thus extends even further than just company to company and even varies from user to user. Therefore, one of the biggest takeaways of this project is that scalability does not have to come with a sacrifice of personalization in corporate wellness applications.

Throughout the development process, the Medella business team team has adhered to agile principles, holding standups every class period and holding a bi-weekly meeting with the client. Using these agile methodologies has allowed us as a development team to keep up with a changing customer vision. An initial set of requirements was found over the course of several meetings with the customer but as time has gone on this has shifted. Even into this spring

semester the customer has mentioned possible shifts in the direction of the project that would dramatically change the line of business Medella would eventually be placed into upon release. Rather than starting over, we have been attempting to set the product up for this transition for whomever begins work on it next. Using the agile development method made this all possible.

1. Introduction

1.1 Problem Statement

Healthcare and insurance are major costs for American businesses. In 2019, the average cost of a company-provided family insurance plan surpassed $20,000 a year, and continues to rise (Matthews, 2019). Currently, our customer is a consultant for various companies, for which he manually creates and tracks content. People in these companies must be manually added to email lists, which can take up to 5 minutes. Creating newsletters is also very time consuming, as the customer must find content and cut/paste into a template; this process can take more than an hour to complete. Since the number of companies our customer can manually work with is limited, he does most of his tracking in a spreadsheet.

Our new web application, Medella, is a health and wellness platform aimed towards creating a standardized way of solving this problem of healthcare costs through education. Through sending users health and wellness educational content, Medella will help these users lead healthier lifestyles, which in turn will help businesses spend less on the healthcare of their employees. By creating a standardized method for adding businesses and users and for creating content and newsletters, we hope to vastly increase the scalability of our customer's past work. We also plan to make it easier for our customer to track businesses and content by adding a dashboard to the website.

1.2 Contributions

We succeeded in building a web application which acts as a platform to educate employees on caring for their health. This project was proposed by our client who works as a consultant with human resources departments for businesses. We created an entirely new tool

which our client and registered businesses can use to sign up users who can then be sent quizzes, newsletters, blog posts, and videos by the application's admin team. Other than employees attached to an employer, individual users can also peruse the website to view content and register if they are interested. By building a unified platform, we have made it easier for businesses to spread health awareness and knowledge to their employees.

Users can track their own health knowledge by viewing their scores on the quizzes they have taken and seeing how the scores change over time. This information is also available to businesses, so they can evaluate the health knowledge of their employee base as a whole. Information sent to users can be further tailored by sending content to specific businesses or to specific interest groups that users can subscribe to. This further enables users to customize their health education experience.

2. Related Work

One existing workplace health product is a mobile-first platform called Limeade ONE. Each company can use Limeade's generic platform that uses a gamified approach involving assessments, pulse surveys, and rewards (*Limeade*). While Limeade provides a variety of services, these overcomplicate what started out as a simple corporate wellness application, and thus the platform has a steep learning curve. Users must now spend time learning to navigate the various features. Furthermore, using Limeade requires knowledge of its file-sharing system, Sharepoint. This makes it more complex for businesses to set up and maintain the platform (Desai & S, 2019).

Our product will serve as a platform with which the Medella team, employers, and users can interact. Users can register for the platform through their employer or independently. The Medella team will be able to create blog posts, upload videos, create quizzes, and write newsletters through forms within the application. Quizzes can include questions such as "How many hours of sleep should you get everyday?" and newsletters can include links to previously created content. Each type of content can be tagged with a topic, which users can subscribe to through their profile page. Medella employees will be able to send these posts to users based on their employer or subscribed tags. Medella's advantages over Limeade include its simplicity, as well as the ability to register as an independent user not associated with a business. These features will make Medella an inclusive platform that everyone can use.

3. System Design

Medella is an online platform that aims to educate users about health and wellness through various forms of media. The overall goal of this project is to enable users to think about incorporating healthy habits in their everyday lives by sending out daily newsletters, blogs, quizzes, and videos. There are three different types of users on this platform: Medella administrators, business administrators, and users. Medella administrators create content, monitor engagement from each business, register businesses, and register business administrators. Business administrators register individual users with their corporation, view employee information, and track employee quiz results. Users can view educational content created by Medella administrators, take quizzes, subscribe to different topics, and update their email preferences. These users can be registered either with a business or individually without a business.

Medella was developed using Django, a Python web framework. This framework was chosen because it is widely used and thus has good community support. Django also has many packages that are easily integrated with the project. In addition, most of the team members had experience programming with Django, so using this framework greatly reduced the time needed to learn how to use it. This project also uses the MIT license, which is a permissive free software license. Users can use software with the MIT license free of charge without restriction. In addition, users are free to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the software without restriction. The MIT license also states that there is no warranty for the software.

<u>3.1 System Requirements</u>

Gathering system requirements allows developers and customers to ensure that they have the same expectations for the project. These requirements also serve as a guidance for developers to follow while completing the project. Below is a list of our system requirements:

Minimum Requirements - complete in Fall 2019

- Businesses, employees, and the Medella team should be able to securely log in.

- The Medella team should be able to create quizzes and blog posts.

- Employees should be able to click on quiz links, view questions, submit answers, and receive a score after submission.

- The Medella team should be able to view aggregate and individual results for each quiz

- Businesses should be able to view aggregated results for each quiz.

- The Medella team should be able to input YouTube links into a form to display them in users' content feeds.

- Employees should be able to watch videos through the platform.

- The Medella team should be able to send emails to employees to inform them of relevant content.

- Employees should be able to opt out of receiving all information from Medella.

- Users should be able to view a page with a mission statement and contact information when going to the webpage without a log-in.

Desired and Stretch Requirements - complete in Spring 2020

- The Medella team should be able to use a newsletter builder form to add text, images, and links to health-related topics to newsletters, as well as publish the newsletters to employees.

- Employees should be able to view previous quiz submissions.

- The Medella team should be able to track clicks by users and businesses on the website in order to monitor user engagement.

- Employees should be able to opt out of receiving only one type of content (quizzes, blogs, videos, newsletters etc.)

- The Medella team should be able to send content to users that are subscribed to specific topics or tags.

3.2 Wireframes

Wireframes are critical during the initial phases of web and mobile design because they focus attention on the basic functionality, layout, and navigation for an interface without the complication of incorporating color theory, images, and other detailed content. They also help to improve the communication between developers and clients throughout the design process by identifying design priorities before implementation. Many of our wireframes have changed since first created because of suggestions and new information from the customer. We have included both the old and new wireframes below.

The landing page is the first page the user will see when they visit the website. It displays Medella's mission statement and the company's contact information.
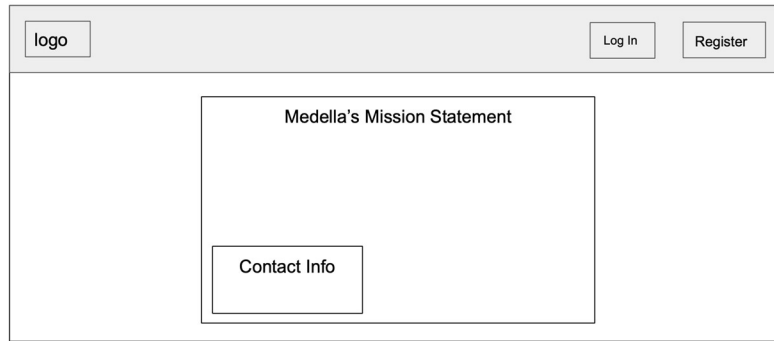
Figure 3.2.1

The login page and register page are built into one view. You have the option to register or login on the same page while viewing the mission statement and contact info.
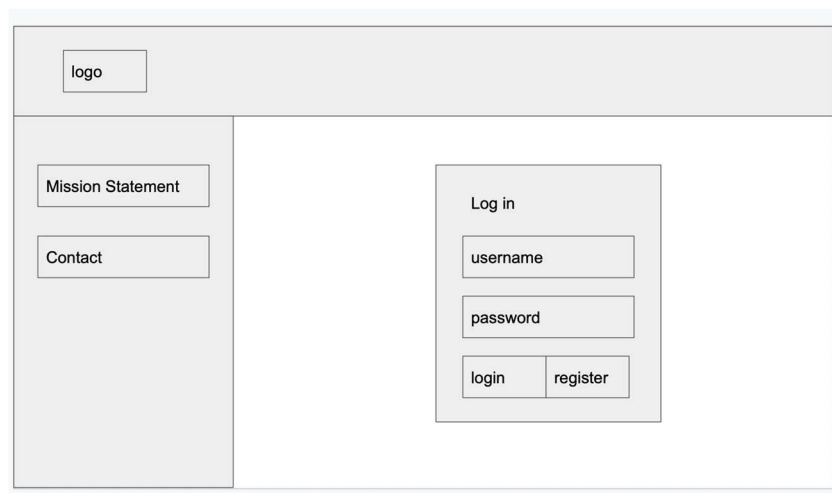

Figure 3.2.2

Originally, when we were planning on receiving more cost data there was a plan to include a company dashboard page. This has since been removed due to difficulties in getting data to be used. Below is the original wireframe.
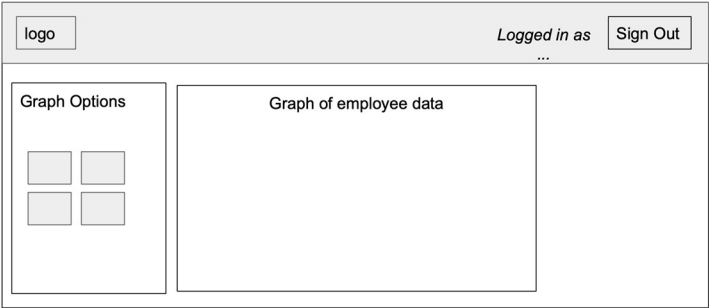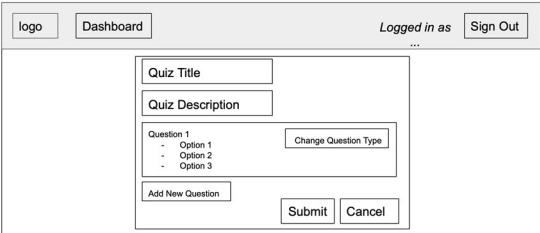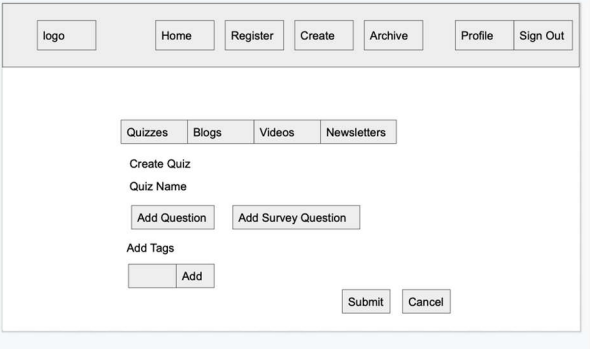
Figure 3.2.3

Our creation feature for quizzes, blogs, newsletters, and videos changed. There is now one "create" button on the navbar which takes users to a tabbed interface where they can select what type of content they want to create. This makes it easier to access other creation forms.

Original Wireframes	Updated Wireframes
Figure 3.2.3	Figure 3.2.4
The quiz creation form (Figure 3.2.4) includes the same elements as before, but is organized in a cleaner fashion. It also has a survey question option and tags.	

Medella Team - Blog Post Creation Form

| logo | Dashboard | | *Logged in as* ... | Sign Out |

| Title | Subject |
| Blog content | |

Submit | Cancel

Categories

Keywords

Figure 3.2.5

| logo | Home | Register | Create | Archive | Profile | Sign Out |

Quizzes | Blogs | Videos | Newsletters

Create Blog

Blog Title | Blog Subject

Blog Content

Add Tags

| Add |

Submit | Cancel

Figure 3.2.6

The new blog design removed Categories and Keywords and replaced them with Tags. (Figure 3.2.6)

Medella Team - Video Creation Form

| logo | Dashboard | | *Logged in as* ... | Sign Out |

Insert Link to Video

Video Preview

Submit | Cancel

Categories

Keywords

Figure 3.2.7

| logo | Home | Register | Create | Archive | Profile | Sign Out |

Quizzes | Blogs | Videos | Newsletters

Create Video

Title | URL

Add Tags

| Add |

Submit | Cancel

Figure 3.2.8

The new video design (Figure 3.2.8) removed Categories and Keywords and replaced them with Tags. We also removed the video preview and added a Title field.

Medella Team - Newsletter Creation Form

| logo | Dashboard | | *Logged in as* ... | Sign Out |

Title

Content

Submit | Cancel

Figure 3.2.9

| logo | Home | Register | Create | Archive | Profile | Sign Out |

Click to add text

Quizzes | Blogs | Videos | Newsletters

Create Newsletter

Newsletter Title | Newsletter Subject

Newsletter Content

Submit | Cancel

Add Blog Links
| Add |
Add Video Links
| Add |
Add Quiz
| Add |
Add Tags
| Add |

Figure 3.2.10

The newsletter creation form (Figure 3.2.10) now includes additional input fields. First off, there has been a subject field added like in the blog creation form. There are also drop down menus for adding links to relevant blogs, quizzes, and videos. Finally, we added the ability to add tags to newsletters.

Employee - Take Quiz

Figure 3.2.11

"Take Quiz" (Figure 3.2.11) remains the same since we are still using TypeForm. A TypeForm quiz that has been created is directly embedded into the page.

3.3 Sample Code

Part of our project involves working with the Typeform platform to build and host quizzes for users. Typeform is a software for building online forms and surveys which can be embedded in web pages. It comes with an API that allows users to construct Typeform forms autonomously by sending JSONs to certain URL endpoints. For our project, we need to take input from a custom-built form, construct the proper JSON, and send the JSON to Typeform in an API call. We wrote a wrapper tool to make this easier.

The code to create the API call from the input:

```
# receiving POST request from quiz builder
elif request.method == 'POST':
    # start constructing the JSON for the Typeform API request
    has_nonsurvey = False
    new_typeform = TypeformQuizBuilder(request.POST['title'])
    # Obtain tags to append to the quiz
    tags = request.POST.getlist('tags[]')

    count = 1
    for item in request.POST:
        # parse the question data sent
        if item == 'questions':
            data = json.loads(request.POST[item])
            # iterate through questions from the form
            for question in data:
                new_typeform.add_question(question, count)
                count += 1

    # own thank you screen as a statement question
    # allows nonsurvey question correct answer explanation to always have a next jump
    new_typeform.add_thankyouscreen(count)
```

Figure 3.3.1

The code shown in Figure 3.3.1 makes use of the wrapper class `TypeFormBuilder` and its

method `add_question`, which appends a new question to the JSON it is constructing for the

API call.

```python
def add_question(self, question, question_count):
    '''
    Add a new question to the JSON for the API call.

    question: a specifically created JSON, returned from create_question
    question_count: variable indicating which question the JSON is for
    '''

    # create explanation text
    explanation_text = ''

    # check if question is a survey type question
    is_nonsurvey = not question[1]

    if is_nonsurvey:
        self.has_nonsurvey = True
        explanation_text = question[3]

    # construct question json
    new_question_json = self.create_question(question[0], question_count)
    answer_count = 1

    # add the answer choices to the question json
    for answer_ind in range(2, len(question)):
        # skip the explanation on nonsurvey questions (False means nonsurvey)
        if(is_nonsurvey and answer_ind != 3) or not is_nonsurvey:
            self.add_answer_choice(new_question_json, question[answer_ind], answer_count)
            answer_count += 1

    # add the question json to the API call
    self.data['fields'].append(new_question_json)
```

Figure 3.3.2

```python
    # add jsons to API call for right/wrong answers
    if is_nonsurvey:
        score_action = self.create_score_action(question_count)
        correct_statement = self.create_correct_statement(question_count, explanation_text)
        incorrect_statement = self.create_incorrect_statement(question_count, explanation_text)
        correct_action = self.create_correct_action(question_count)
        incorrect_action = self.create_incorrect_action(question_count)
        correct_to_next_action = self.create_correct_to_next_action(question_count)

        self.data['fields'].append(correct_statement)
        self.data['fields'].append(incorrect_statement)
        self.data['logic'].append({
            'actions': [score_action, correct_action, incorrect_action],
            'type': 'field',
            'ref': 'question_' + str(question_count)
        })
        self.data['logic'].append({
            'actions': [correct_to_next_action],
            'type': 'field',
            'ref': 'correct_statement_'+ str(question_count)
        })
```

Figure 3.3.3

The code for `add_question` is shown in Figures 3.3.2-3.3.3.

3.4 Sample Tests

Testing allows software to be proven functional. 100% code coverage cannot guarantee a system will be bug free, but tests will be able to prevent some bugs and thus prove some level of adequacy in the system. Extensive testing gives a sense of security that when using this platform as it shows that work has been done to ensure it is designed well. The system has not been thrown together haphazardly with no regard for how it will actually function. On top of this, writing tests makes developers think about all the ways that their code could possibly be broken. It forces consideration of all the different conditional paths through the system which can lead to bugs being found before they become a problem in production.

The reason that this last piece is important is the drastic increase in cost associated with fixing bugs in production as opposed to development. Implementing solutions after a project has been put into production will be exponentially more expensive than implementing a solution during development. Therefore, testing should be done early and often in an attempt to prevent problems later on in the software's lifecycle. Clients who are aware of this idea will also be much more comfortable paying for software that has been thoroughly tested. It gives a sense of security for their own investments in the long term.

In the remaining part of this subsection, we provide some sample tests from our testing suite followed by brief explanations of their function.

```
64          '''Tests that a Medella admin can be registered properly with a new Medella account'''
65      def test_new_bus_admin(self):
66          client = Client()
67          client.login(username='medella', password='medella')
68          POST_data1 = {
69              'medella_email': 'user1@email.com'
70          }
71          response = client.post(reverse('main:medella_admin/register_medella_admin'), POST_data1)
72          rt = RegistrationToken.objects.get(email='user1@email.com', access_level=4)
73          POST_data2 = {
74              'first_name': 'User',
75              'last_name': '1',
76              'username': 'user1',
77              'email': 'user1@email.com',
78              'password': 'testuser1',
79              'business': 'Medella',
80              'phone_number': '1112223333'
81          }
82          medella = Business.objects.get(company_name="Medella")
83          token_id = rt.token
84          response = client.post(reverse('main:register', args=(token_id,)), POST_data2)
85          self.assertTrue(Employee.objects.filter(email='user1@email.com', employer=medella, access_level=4))
```

Figure 3.4.1

This test shown in Figure 3.4.1 creates a business admin for the business "Medella".

Again, the first thing that has to be done is log-in as a medella admin (lines 67-68). After that a

registration token is generated for the user to use to become a business admin (lines 71-72).

Typically this would be a link sent out in an email. However, for testing purposes it is instead

just pulled from the database. Then, the POST request of info that would be input by the new

user for creating their new business admin account is created (lines 73-81) and finally submitted

(line 84). The url is generated using the token that was grabbed from the database before hand.

The last statement is used to make sure that a business admin, defined by access level 4, actually

exists in the database with the information passed in through the previous POST request.

```
458        def test_medella_view_nonexistent_quiz(self):
459            """
460            Ensures that when you view a quiz that doesn't exist, you get the proper error message
461            """
462            admin = User.objects.get(username='admin')
463            request = HttpRequest()
464            request.user = admin
465
466            quiz_id = 'fake_id'
467
468            response = medella_view_quiz(request, quiz_id)
469
470            # Retrieve source HTML code from page
471            html = response.content.decode('utf8')
472
473            error_message = "Invalid Quiz ID"
474
475            self.assertTrue(error_message in html)
```

Figure 3.4.2

Figure 3.4.2 shows a test case that looks at the functionality regarding viewing a specific

quiz embedded in the web page. Specifically, for a quiz with an ID not recognized in the

database, the page should display an error message. Lines 462-464 are finding an admin user and

constructing an HTTP request with the user. Line 466 creates a fake quiz ID that will not be

recognized in the database. Lines 468-475 make the response to the quiz page, retrieve the

resulting page HTML, and verify that the error message appears in the HTML.

```
182         ''' Makes sure employee sees what has been dispatched to them in the quiz archive.
183         Doesn't use built-in sending functionality. Simulates it by adding entries directly to
184         the dispatch table'''
185     def test_employee_quiz_archive_one_sent(self):
186         client = Client()
187         client.login(username='user', password='user')
188         user = User.objects.get(username='user')
189
190         # Simulate Dispatch (type 1 represents video)
191         curr_quiz = Quiz.objects.get(title="Quiz")
192         Dispatch.objects.create(type=3, content_id=curr_quiz.id, user=user)
193
194         sent_quiz_ids = Dispatch.objects.filter(type=3, user=user).values_list('content_id')
195         sent_quizzes = Quiz.objects.filter(id__in=sent_quiz_ids)
196         response = client.get(reverse('main:employee/quiz_archive'))
197
198         self.assertEqual(CORRECT_CODE, response.status_code)
199         self.assertEqual(set(sent_quizzes), set(response.context['quizzes']))
```

Figure 3.4.3

The test shown in Figure 3.4.3 is used in order to ensure content will be made visible to employees in their own archive page after it has been sent to them. To do this, first the dispatch process is simulated in lines 191-192. That is, rather than using the send email functionality of the application, an entry is manually inserted into the table responsible for tracking what has been sent out to different users. Then, when the response for loading the archive is received it is ensured that only the quiz that has been sent to the current user is loaded on line 199.

3.5 Code Coverage

In order to perform code coverage as a team we have used coverage.py. This is designed for python programs and is a simple python package that can be installed in one of two ways. Either using `pip install coverage` or using `sudo apt-get install python-coverage` depending on what OS you are running. From there, all that is necessary is to run the testing suite using the newly added command `coverage run`. Thus, the complete command for creating a code coverage report will be `coverage run manage.py test`. Once that completes, a basic report can be generated in the terminal using `coverage report`

or a more detailed report displaying for each line whether or not it is included in a test using `coverage html.` This second option will create a folder containing an index.html file that will allow for easy viewing of every file in the project and its current code coverage. Using this tool, coverage of the project is currently at 99%.

3.6 Installation Instructions

How to create and configure an AWS Instance

1. Go to aws.amazon.com

2. Register an account with Amazon AWS

3. Log into the AWS console and provision an EC2 instance

    a. In Step 1 (Choose AMI), search for "Ubuntu Server 18.04 LTS (HVM), SSD Volume Type" as the AMI. Keep the 64-bit (x86) option selected.

    b. In Step 2 (Choose Instance Type), select at least a T2-small instance (you may have to use a larger instance depending on the number of users)

    c. In Step 3 (Configure Instance Details), check *Protect against accidental termination*

    d. In Step 4 (Add Storage), change the Size to at least 30 GiB

    e. In Step 6 (Configure Security Group), keep "create a new security group" selected. You may want to change the name of the security group to something you can remember. Make a note of the Security Group name

    f. Click Review and Launch, and Launch the Instance

    g. Create a new key pair and name it

        i.   Download the Key Pair and make a note of the file location. You will need

to remember the file location to access the VM later.

4. While you wait for the instance to boot, go back to the EC2 homepage and click

"Security Groups" which should be located in the "Resources" box. Go to the security

group you just created and add the following inbound rules:

    a. HTTP for port 80

    b. Custom TCP Rule for port 8000 (for testing only, could be deleted after

installation is complete)

    c. HTTPS for port 443

    d. SSH for port 22 (this one should already be there)

5. Click save

How to connect to the AWS Instance

1. Go to the EC2 homepage and click "Running Instances." If the status checks says "2/2

checks", you can connect. If it still says "initializing," wait until the status changes to

"2/2 checks."

2. Open the command line

    a. On Windows, type `cmd` into the Start Menu

    b. On MacOS, open the Terminal

3. Navigate to the directory where you downloaded the Key Pair

4. In the browser, right click on the Instance in the AWS Console, click *Connect*, and follow

the instructions given on the page to connect to the instance

5. On the "Connect" page, make a note of your public DNS. It should look something like

   ec2-3-20-161-168.us-east-2.compute.amazonaws.com

How to set up mysql on the EC2 Instance

1. Connect to the EC2 instance

2. Run `sudo apt-get update`

3. Run `sudo apt-get install mysql-server`

4. Log into the mysql server by running `sudo mysql -u root`

5. Update the password for the root (most privileged user) by running `ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'new-password';` where *new-password* is the new password

6. Exit the mysql terminal by running `quit`

7. Restart the service by running `sudo service mysql restart`

8. Log into the mysql server again by running `mysql -u root -p`

   a. It will prompt you for the password you just set

9. Create a new user to use in the Django app. We are using 'medella' for the username and 'capstone' for the password. Run `CREATE USER 'medella'@'localhost' IDENTIFIED BY 'capstone';`

10. Grant all privileges to the newly created user by running `GRANT ALL PRIVILEGES ON * . * TO 'medella'@'localhost';`

11. Flush the privileges for the changes to take effect by running `FLUSH PRIVILEGES;`

12. Exit the mysql terminal by running `quit`

13. Log into the mysql server with the new user account `mysql -u medella -p`. It will

    prompt you for the password you set earlier (capstone)

14. Create a new database for the Django app to use by running `CREATE DATABASE`

    `medella_db;`

    a. Note: the name of the database must be 'medella_db' in order for the Django app

       to correctly connect to the database backend. You could also change this in the

       settings.py file of the Django app for a new database name.

15. Because our Django app stores strings in UTF-8 instead of the default format specified by

    MySQL, change the character set of the database by running `ALTER DATABASE`

    `medella_db CHARACTER SET utf8 COLLATE utf8_general_ci;`

16. Exit the mysql terminal by running `quit`

17. Install the mysql-config package by running `sudo apt-get install`

    `libmysqlclient-dev`

How to install the Django app on the EC2 Instance

1. Connect to the EC2 instance

2. Open the terminal and run `sudo apt-get update`

3. Then run `sudo apt-get install python3-pip`

4. Run `sudo pip3 install --upgrade pip`

5. Run `wget --no-check-certificate`

   `'https://docs.google.com/uc?export=download&id=1mowMydljuqB`

   `Elsbih236gLRAcBFJk88o' -O medella.zip`

a. Alternatively, you can download the zip file from

https://drive.google.com/open?id=1mowMydljuqBElsbih236gLRAcBFJk88o and

transfer it to the VM if you wish

6. Unzip the zip file

a. Run `sudo apt install unzip`

b. Run `unzip medella.zip`

7. Run `cd medella/`

8. Run `cd src/`

9. Run `pip3 install -r requirements-travis.txt`

10. Run `cd medella/`

11. Open settings.py by running `nano settings.py`

12. Scroll to the ALLOWED_HOSTS (should be around line 28)

a. Add your EC2's public DNS to this list (this can be found when you right click

on your instance in the AWS console and click "Connect")

b. Save (Ctrl-X, y, enter)

13. Run `cd ..` (navigate back into medella/src/)

14. Run `python3 manage.py migrate`

15. Run `python3 manage.py createsuperuser`

a. Use whatever username and password you like, we use "admin" and "admin"

b. Run `./runserver.sh` and the website will start running in approximately 1

minute

i. If you get an error that says "Permission denied", run `chmod 777`

`runserver.sh` to change the permissions before running the executable

    c. Go to `<public DNS>`:8000, in our case it is http://ec2-3-20-161-168.us-east-2.compute.amazonaws.com:8000/

How to set up a Typeform API key and integrate it into the service

1. Install and set up the project

2. Go to https://www.typeform.com/

3. Click "Sign up" to register an account

4. Log in at https://developer.typeform.com/

5. In the upper right hand corner, click "Settings"

6. On the left, under "Profile," click "Personal tokens" and click the option "Generate a new token," use full scope

7. The character string returned is the API key. Go to tokens.py (medella/src/main/tokens.py) in the installed project and set the variable `typeform_token` to the API key

8. Note that Typeform's free tier has a limited number of API requests per month. If you are a student, you can use the [GitHub Student Developer Pack](https://github.com) in order to get a free 1-year trial of Typeform's professional tier. It may take a few days for verification to process.

How to set up a Youtube API key and integrate it into the service

1. Install and set up the project

2. Go to https://console.developers.google.com/apis/credentials and sign in with your Google account

3. Click "Create a project" on the right hand side, and then click "Create" with the default project options

4. Then, at the top of the page click "Create Credentials," and then click "API Key"

5. Copy this key, and then navigate to tokens.py (medella/src/main/tokens.py) in the installed project

6. Set the variable `youtube_token` to the copied key

4. Results

The resulting system met all of the customer's minimum and desired requirements. Beforehand, the customer did not have an automated way to consult with businesses to educate the health of their employees and had to perform those tasks by hand. The automated system allows multiple users to register themselves on the service, and the Medella admin user will be able to disperse content to many users at once. This offers a big advantage in scale, as the customer can reach a large audience much more easily and multiple companies can onboard themselves into the service at once. Other stakeholders, such as businesses that register for the service, will be able to keep their employees educated about their health through access to the content sent out by the Medella admins instead of finding the content themselves. They can also have Medella admins send their employees quizzes on topics that the business wants to educate its employees about. On the employee users' end, they will be able to follow tags that customize the kinds of content that users are more interested in. This allows them to tailor content to their specific wellness needs. They can also use quizzes as a metric to gauge their wellness knowledge.

In terms of the functionality itself, the customer will be able to carry out tasks more efficiently. For example, one of our key features is the ability to create a newsletter and to send it to users. Our customer's old process consisted of copying and pasting content into a template and emailing it out manually, which can take anywhere from 30 to 60 minutes. With the new process, the customer builds the newsletter on our platform and clicks a button to send it to all users following a specific tag or business. At most, this will take 20 minutes to achieve.

In terms of tracking the users engagement with the sent out content, this was never able to be accurately tracked until now. Since our customer's old process consisted of manually

emailing newsletters, blogs, etc., there was no way to check who was actually reading which emails. With the system we have built, our customer can now view breakdowns of the engagement with different kinds of content Medella will send out, allowing the company to figure out which kinds of content the users prefer. This information is now available at a glance by viewing the home dashboard, and provides far more feedback than sending out emails could. The dashboard breaks engagement down by company as well as the type of content (quiz, blog, and video), allowing our customer to see which companies prefer which type of content. In the future, the engagement will also be broken down by category, providing a more detailed dashboard and allowing the content that is being sent out to be more finely tuned.

5. Conclusions

Building the Medella platform showed us the power of technology in impacting people's lives on a larger scale. Before Medella, our customer was attempting to decrease health insurance costs on his own. By working with smaller companies, our customer would send wellness information to individual employees, which was time-consuming and tedious. With our new web application, our customer will have a standardized way of creating and distributing this information, exponentially increasing the impact of his work. Our customer will also have a simpler way to see consolidated information of all of the companies he is helping, which will increase efficiency in deciding where to focus his efforts. Hopefully, with the ability to access an increased number of people and companies, our customer will be able to help improve the health and decrease the health insurance costs for a number of people across the country.

6. Future Work

Future work on this project may include health insurance cost tracking for businesses. Business administrators should be able to track the costs of certain healthcare areas (such as diabetes, cardiology, etc.) through the company's health insurance. These administrators can choose to send content about these topics to their employees. Over time, business administrators can see the changes in insurance costs as a measure of the impact that Medella has had on their employees' well-being.

Additionally, this project may be expanded by tailoring content to each individual user's interests in more refined ways. Currently, Medella subscribes users to a tag if they view related content at least once. However, this is not the most accurate way to gauge a user's interest in a topic. Future work on Medella may include using more complex algorithms to determine a specific user's interests. This may encourage users to use Medella's online platform more frequently when they see more content that appeals to them.

7. References

Desai, N., & S, A. (2019, February 20). Limeade ONE Reviews 2019. Retrieved from

g2.com/products/limeade-one/reviews.

Limeade. (n.d.). Retrieved October 11, 2019, from https://limeade.com/.

Mathews, A. W. (2019, September 25). Cost of Employer-Provided Health Coverage Passes

$20,000 a Year. *Wall Street Journal*. https://www.wsj.com/articles/cost-of-employer-

provided-health-coverage-passes-20-000-a-year-11569429000