Identyti: Digital Storage of Personal Identification Documents

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

Eric Burbach
Spring, 2020

Technical Project Team Members
Eric Burbach
Chris Han
Sri Jayakumar
Samantha Kostleni
Gio Lee
Amanda Murray

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

Identyti: Digital Storage of Personal Identification Documents

Table of Contents

Abstract 2

List of Figures 4

1. Introduction 6

1.1 Problem Statement 6

1.2 Contributions 8

2. Related Work 9

3. System Design 10

3.1 System Requirements 11

3.2 Wireframes 13

3.3 Sample Code 14

3.3.1 Models 14

3.3.2 Views 17

3.3.3 Forms 22

3.3.4 ML Model 23

3.4 Sample Tests 25

3.5 Code Coverage 26

3.6 Installation Instructions 26

3.6.1 Setting up the Repo: 27

3.6.2 Deploying with Heroku: 35

3.6.3 Configure the Database: 40

3.6.4 Configure OAuth: 45

3.6.5 Setting up AWS 52

3.6.6 Setting up email with SendGrid 63

3.6.7 Setting up Tesseract for local testing 66

4. Results 68

5. Conclusions 69

6. Future Work 70

7. References 71

Abstract

The purpose of the Identyti capstone project was to build a proof-of-concept for a business idea that Darden School of Business student, James Henderson, had. The idea is that all personal identity documents can be stored safely online in a centralized location where they can be accessed by anyone who needs them. Possible use cases for the application would be applying for a driver's license, filling out and providing documents for a W-4 form, and filling out insurance paperwork at a doctor's office. Identyti would make these processes easier for both the customer and the organization involved by eliminating the need to keep hard copies of paperwork.

The proof of concept was built as a web application using the Django web framework with the important documents stored on Amazon Web Services Simple Storage Service. Some other libraries used in the application were Google OAuth, SendGrid, and Tesseract OCR. Some unique features of the app include secure storage of the documents, mobile responsiveness, and a Machine Learning model used to categorize the document being uploaded as either Identification, Insurance, Financial, Medical, Academic/Work, or Legal. The most important feature that this app requires is security of a user's documents. This key feature is what separates Identyti from other digital storage applications currently available.

The product delivered to Henderson was a web application with the desired requirements specified at the beginning of the project completed and fully functional. The purpose of this product is to give Henderson a proof-of-concept that he can test with customers in order to get

information on the viability of his business. The data collected by Henderson is out of the scope of this capstone project.

List of Figures

Figure 1: Wireframe of Homepage
Figure 2: Wireframe of Category Page
Figure 3: Github homepage
Figure 4: Creating a github repo
Figure 5: Your repositories list on github
Figure 6: Screenshot of setting up repo 1
Figure 7: Screenshot of setting up repo 2
Figure 8: Screenshot of setting up repo 3
Figure 9: Screenshot of setting up repo 4
Figure 10: Screenshot of setting up repo 5
Figure 11: Screenshot of deploying with Heroku 1
Figure 12: Screenshot of deploying with Heroku 2
Figure 13: Screenshot of deploying with Heroku 3
Figure 14: Screenshot of deploying with Heroku 4
Figure 15: Screenshot of deploying with Heroku 5
Figure 16: Screenshot of deploying with Heroku 6
Figure 17: Screenshot of deploying with Heroku 7
Figure 18: Screenshot of configuring the database 1
Figure 19: Screenshot of configuring the database 2
Figure 20: Screenshot of configuring the database 3
Figure 21: Screenshot of configuring the database 4
Figure 22: Screenshot of configuring the database 5
Figure 23: Screenshot of configuring the database 6
Figure 24: Screenshot of configuring the database 7
Figure 25: Screenshot of configuring OAuth 1
Figure 26: Screenshot of configuring OAuth 2
Figure 27: Screenshot of configuring OAuth 3
Figure 28: Screenshot of configuring OAuth 4
Figure 29: Screenshot of configuring OAuth 5
Figure 30: Screenshot of configuring OAuth 6
Figure 31: Screenshot of configuring OAuth 7
Figure 32: Screenshot of configuring OAuth 8
Figure 33: Screenshot of configuring OAuth 9
Figure 34: Screenshot of configuring OAuth 10
Figure 35: Screenshot of configuring OAuth 11
Figure 36: Screenshot of setting up AWS 1
Figure 37: Screenshot of setting up AWS 2

Figure 38: Screenshot of setting up AWS 3
Figure 39: Screenshot of setting up AWS 4
Figure 40: Screenshot of setting up AWS 5
Figure 41: Screenshot of setting up AWS 6
Figure 42: Screenshot of setting up AWS 7
Figure 43: Screenshot of setting up AWS 8
Figure 44: Screenshot of setting up AWS 9
Figure 45: Screenshot of setting up AWS 10
Figure 46: Screenshot of setting up AWS 11
Figure 47: Screenshot of setting up AWS 12
Figure 48: Screenshot of setting up AWS 13
Figure 49: Screenshot of setting up AWS 14
Figure 50: Screenshot of setting up AWS 15
Figure 51: Screenshot of setting up email with SendGrid 1
Figure 52: Screenshot of setting up email with SendGrid 2
Figure 53: Screenshot of setting up email with SendGrid 3

1. Introduction

The average wait time at the DMV is 45 minutes, most doctor's appointments start with filling out paperwork, and most companies require new employees to come in to present their Passport before they can start working. These are all just a few examples of places where Americans waste time dealing with paper copies of personal identification documents. The internet could be used to host all of these documents and save people time and money.

1.1 Problem Statement

Identyti is a Darden School of Business start-up seeking to change the way the world shares and receives identification documents. Currently, most identification documents (drivers license, passport, insurance card, etc) are handled as strictly hard copies, and the process for getting those documents can be difficult.

While there are currently some ways to verify the validity of someone's driver's license online, those are predominately used for the sale of alcohol and other restricted substances and not for any official government identification (Larson, 2019). This means that people must keep hard copies of these documents and all the documents they need to apply for them, which can be burdensome. One could digitize their own documents, but that leads to another problem; there is no system designed for digital storage of sensitive personal documents. WIRED magazine recently published a "Guide to Your Personal Data (and Who Is Using It)" in which they detailed how companies like Google are collecting and storing every click and keystroke made in their web browser, store this data along with user identities, and use it to serve targeted ads to the

users (Matsakis, 2019). In other words, major document storage companies (Google, Dropbox) collect data on their users and leverage that data for capital gain. Storing private documents on those servers makes users vulnerable.

This means people have no choice but to retain hard copies of all of their personal documents which can be difficult for those who move around a lot (i.e., college students and those in the military). Simple tasks like going to the DMV are so complicated that there are articles online like 'How to Get Through the DMV with your Sanity Intact' that include steps like 'Gather All the Paperwork You Need' highlighting the fact that both going to the DMV and gathering paperwork are much more difficult than they need to be (Klosowski 2013). The Identyti system will be beneficial because it will allow for the simplification of processes such as renewing a driver's license, giving people back the free time to focus on what is important in life.

Identyti also provides benefits for enterprise clients. A major problem that large enterprises face is the sheer size of information they have to process and handle. Using Identyti, enterprises have the ability to create an account in order to manage all of their employees' data. For example, when onboarding a new employee, an enterprise client could request the necessary documents, such as identification and tax reports, from the new employee through Identyti. In this way, Identyti creates a secure and simple path of communication between enterprises and employees for personal, confidential documents. Instead of requiring users to carry physical documents and submit them to enterprises, Identyti creates a channel to share these documents electronically.

1.2 Contributions

The product, created as part of this capstone project, allows users to upload personal documents to the site and access them whenever needed; the site also allows enterprises to create accounts and access documents from users with their approval. The rest of this thesis is organized into four sections. In Section 2, the current options for online document storage are discussed. Section 3 shows our approach to address the aforementioned problem stated, our web-based application, as well as the system design. The results of our work are discussed in Section 4. Finally, Section 5 concludes this thesis and section 6 discusses the future of the work performed for this class and how Henderson may use the application built.

2. Related Work

There are file storage systems that already exist, like Google Drive and Dropbox. Those systems have carved out a niche for themselves in the world of sharing files in order to work on them concurrently. These systems, that are already in place, do not focus on storage of important and personal documents like Identyti. This is a key distinction, where an important gap in electronic file storage systems can be filled by Identyti. Current file storage systems do not have the ability to securely and conveniently organize and store personal documents with the intention to share those files with the necessary parties when necessary. As more and more of daily life is overtaken by technology, there is a growing need for personal documents to be stored electronically. Employee onboarding and DMV applications, for example, can all happen online now. Normally, these processes are convoluted and vary greatly from case to case. When these processes are ported online, they are made much more straightforward and case-specific very easily. With this conversion growing in popularity, it is also important to have the ability to access and share personal documents online to complete these processes. Currently, there is no system in place that can handle these specific use cases with ease. Identyti provides a clean and obvious solution to this problem.

3. System Design

Identyti seeks to help users keep their private documents private, and to make everyone's life a little bit easier. Our deliverable is a proof-of-concept showing documents that can be stored securely on the internet, which will be used when meeting with investors and potential stakeholders to garner support for this technology. As such, the fundamental goal of our project was to build a document storage service that is secure, easy to navigate, and allows users to access the product on mobile and desktop devices. We accomplished this by creating a Django application that had a flexible user interface for being accessed on mobile devices. Users are able to upload, view, and download their documents on demand. This product also supports enterprise accounts which can issue personal documents and tell users what they need to have to apply for such documents.

Because the documents uploaded to this platform are typically highly sensitive, data security was a critical challenge to tackle. Login authentication is performed by Auth0 which already provides an array of mitigating web-app security tools such as anomaly detection and force email verification (Poza 2018). Additionally, in order to allow clients to access their data from anywhere, the documents they upload will be stored in the cloud, specifically in an AWS S3 bucket. When uploading the documents, the application will save certain metadata associated with the document allowing the client to easily search/sort/categorize the document, making it convenient and faster to find it later. The International Data Corporation, is a provider of market intelligence, conducted a study on their workers to gauge how much time they spend weekly looking for physical documents. In a group of 1200 workers, IDC found that "they spend an

average of 4.5 hours a week looking for documents" (Biddle, 2017). Since Identyti is targeted at both consumers and enterprise clients alike, searching for these documents on Identyti, whether the client is an individual or a business, will be much faster than searching for physical documents.

One concern for storing sensitive data in the cloud is that these services can be compromised. However, in order to mitigate this issue, data stored in S3 is encrypted. This ensures that even if the bucket is compromised, only encrypted data can be recovered. By storing these encrypted files in the cloud, Identyti offers a secure and fast solution for clients, allowing them to easily share their documents with enterprises.

3.1 System Requirements

Requirements were very important in our design of this product because they helped us communicate with our customer and ensure that we were on the same page about what needed to be developed. They also helped us communicate as a team to ensure that we were all working towards the same goals; they were also useful for dividing up the work on system features.

To fulfill our goals we've identified these requirements:

Minimum (Completed by the end of Fall 2019)

- As a user, I should be able to create a customer account, so that I can access persisted data.
- As a user, I should be able to login to my account, so that I can access persisted data.
- As a user, I would like to be able to upload my documents, so that I can store them for future use.

- As a user, I would like my documents to be stored, so that I can access them for future use.

- As a user, I should be able to view my documents, so that I feel confident my documents are stored.

- As a user, I would only like my documents to be accessed by those I have authorized, so that I feel confident in the security of my personal information.

- As a user, I would like to be able to easily locate my documents, so that I can find what I need when I need it.

- As a user, I should be able to retrieve my documents, so that I can use them when I need them.

- As a user, I should be able to access the application from a mobile device, so that I can upload or share documents on the go.

Desired (Completed by the end of Spring 2020)

- As a user, I should be able to search for my documents, so that I can easily find documents when I need them.

- As a user, I should have different options for sorting and filtering my documents, so that I can easily find relevant documents when I need them.

- As a user, I would like the app to recognize what type of document I am uploading (identification, medical, etc) , so that it is easy for me to organize and find my documents later.

- As a user, I should be able to create an enterprise account.

- As a user, I should be able to authorize other users to view my documents, so that I can provide my documents when necessary.

- As an enterprise client, I should be able to request documents from other users, so that I can validate their identity and provide access to appropriate resources.

Optional Requirements (not completed)

- As a user, I should be able to limit the time another user can view my document

- As an enterprise client, I should be able to create a new type of document with a list of required documents needed

- As a user, I should be able to identify which required documents I am missing

- As a user I should be able to delete my documents
- As an admin, I should not be able to view user's documents

3.2 Wireframes

Wireframes were another important tool in communicating with our client. Our client had a background in Product Management at a tech company and already had an image in mind for the product, so he provided wireframes for us. Below are wireframes he gave us for the customer views.

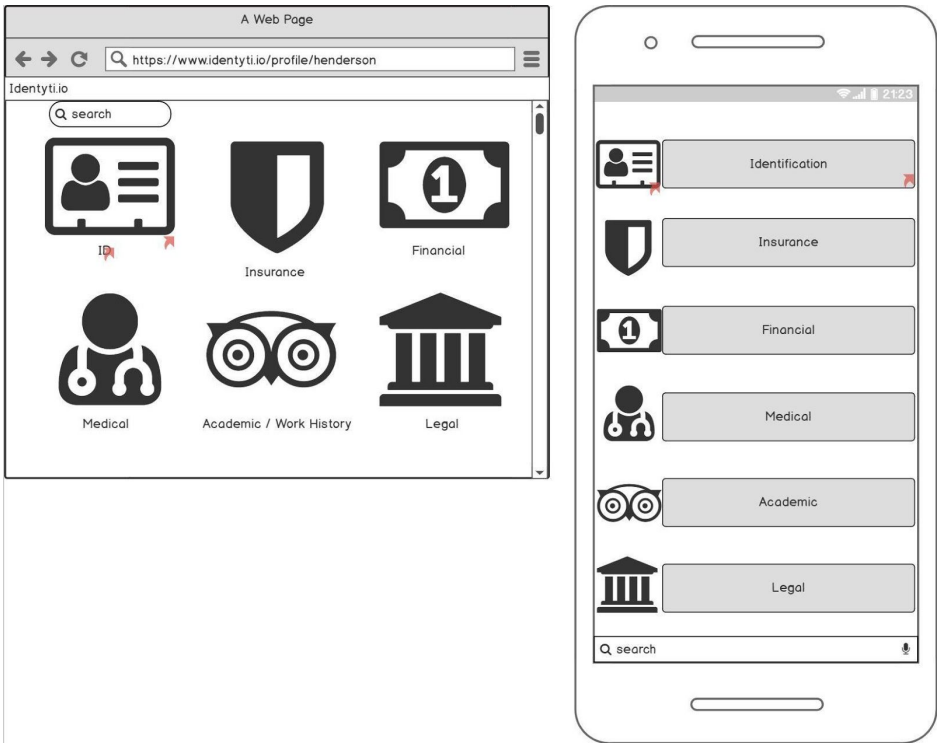


Figure 1

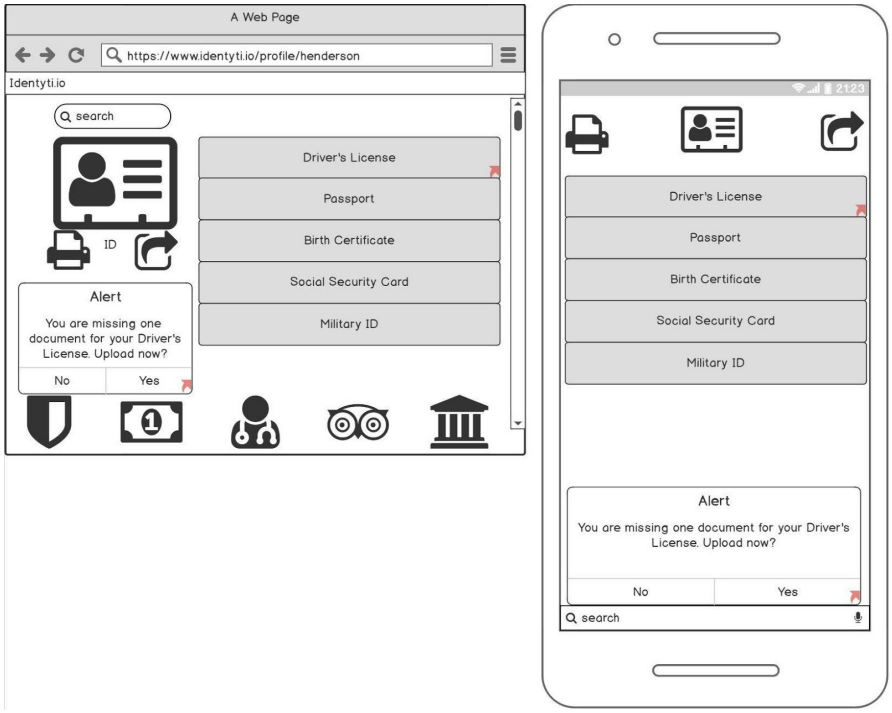


Figure 2

3.3 Sample Code

3.3.1 Models

Each model corresponds to a table in the database.

```
#database entry for enterprise client

class Enterprise(models.Model):

    name = models.TextField(primary_key=True)

    code = models.TextField(blank=True)

    email_contact = models.EmailField() # primary contact for the client
```

The enterprise model contains a field for the name of the enterprise, a unique code that enterprise can use to sign up (stored as a sha256 hash), and an email for the primary contact at that company.

```
class Document(models.Model):

    document = models.FileField(upload_to=user_directory_path,
storage=MediaStorage())

    datetime = models.DateTimeField(default=now)

    name = models.CharField(max_length=256, default='xxxx')

    size = models.IntegerField(default=-1)

    IDENTIFICATION = 'ID'

    LEGAL = "LG"

    MEDICAL = "MD"

    INSURANCE = "IN"

    FINANCIAL = "FN"

    WORK_ACADEMIC = "WA"

    TYPEOF_DOCUMENT_CHOICES = (

        (IDENTIFICATION, 'Identification'),

        (LEGAL, 'Legal'),

        (MEDICAL, 'Medical'),

        (INSURANCE, 'Insurance'),

        (FINANCIAL, 'Financial'),

        (WORK_ACADEMIC, 'Work & Academic'),

    )

    type_of_document = models.CharField(

        max_length=2,
```

```
        choices=TYPEOF_DOCUMENT_CHOICES,
        default=IDENTIFICATION,
    )
    owner = models.ForeignKey(Profile, on_delete=models.SET_NULL, null=True)
```

The document model stores key information about the document that helps us access it in AWS S3, where it is stored. The main details we store are the owner, type of document, uploaded time, filename, and size. It also has a foreignkey of a profile which is the owner of the document. When saving this model, it uploads the document's content to our specified S3 bucket under the owner's name and type of document.

```
class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    is_enterprise = models.BooleanField(default=False)
    enterprise = models.ForeignKey(Enterprise,
on_delete=models.CASCADE,null=True, blank=True)
```

The profile model has a one to one relationship with the Django user model which we used for authentication. There is also a field that distinguishes whether it is a customer account or an enterprise account. If it is an enterprise account, there will be a foreign key that corresponds to an entry in the enterprises database.

3.3.2 Views

These are some of our more important and more complicated view functions. These function as the controller, which handles all the intense logic, in Django.

```
# Uploading a document from index.html

def upload_file(request):

    current_category = request.POST['current_category']

    if current_category:

        form = DocumentForm(request.POST, request.FILES)

        if form.is_valid():

            document = form.save(commit=False)

            document.user = request.user

            document.owner = Profile.objects.get(user=request.user)

            document.save()

        form = DocumentForm(initial={'type_of_document': current_category})

        docs =
Document.objects.all().filter(owner=Profile.objects.get(user=request.user))

        return render(request, 'identyti_app/category_file_view.html',
{'form': form, 'cat': current_category, "docs": docs})

    if request.method == 'POST':

        form = DocumentForm(request.POST, request.FILES)

        if form.is_valid():

            doc = form.save(commit=False)

            doc.user = request.user

            doc.owner = Profile.objects.get(user=request.user)

            doc.name = doc.document.name
```

```
            doc.size = doc.document.size
            doc.save()
        form = DocumentForm()
    else:
        form = DocumentForm()
    return render(request, 'identyti_app/index.html', {'form': form})
```

The upload_file view first checks for a current category. If there is a current category, this means we must render a current category page rather than the index page after uploading a given file. Once this is determined, the file is taken from the request object. From the request, a Document model is created for the file and the file's info is stored in this model object. The object is assigned to the user who has uploaded it. After this, the response is rendered according to the determination of the current category at the beginning.

```
# Search for documents for a user
def search(request):
     if request.method == 'GET':
        query = request.GET.get('search-bar')
        docs =
Document.objects.all().filter(owner=Profile.objects.get(user=request.user))
        if query != None:
            results = docs.filter(document__icontains=query)
            return render(request, "identyti_app/search.html",
{"results":results})
        # else if filter tags
```

```
        elif "namebtn" in request.GET:

            # show name elements

            results = docs.order_by('name')

            return render(request, "identyti_app/search.html",
{"results":results})

        elif "datebtn" in request.GET:

            # show date elements

            results = docs.order_by('datetime')

            return render(request, "identyti_app/search.html",
{"results":results})

        elif "sizebtn" in request.GET:

            # show size elements

            results = docs.order_by('size')

            return render(request, "identyti_app/search.html",
{"results":results})

        elif "showallbtn" in request.GET:

            # show all elements

            results = docs

            return render(request, "identyti_app/search.html",
{"results":results})

        # Else return blank

        else:

            return render(request, "identyti_app/search.html", {})

    # Incorrect render

    else:

        print("error")

        return render(request, "identyti_app/search.html", {})
```

The search view handles searching and filtering documents. First, the query is taken. If a filter button was clicked on, the query will have the value ‘None.’ If there was a search, the query will be the content of that search. Then the user’s documents are represented by the variable docs. The view checks if there is a search query. If there is, the view returns a filtered list that matches the search query. If there is no search query, the view checks which button was clicked. Whichever button was clicked will sort the docs by the specified button, then return the filtered docs.

```
def enterprise_sign_up(request):
    if request.method == 'POST':
        form = EnterpriseSignUpForm(request.POST)
        if(form.is_valid):
            #Check if the confirmation code is correct
            raw_code = request.POST.get('code','')
            encoded_code = sha256(str(raw_code).encode('utf-8'))
            enterprise = request.POST.get('enterprise', '')
            try:
                actual = Enterprise.objects.get(name = enterprise)
            except ObjectDoesNotExist:
                return render(request,
'identyti_app/enterprise_sign_up.html', {'form': form, 'no_query':True})
            if(encoded_code.hexdigest() == actual.code):
                #Creates a user object and attaches it and an Enterprise to
an EnterpriseProfile
                user = User.objects.create_user(
```

```
                username = request.POST.get('username',''),

                password = request.POST.get('password',''),

                email = request.POST.get('email','')

                )

                user.save()

                profile = user.profile

                profile.is_enterprise = True

                profile.enterprise = actual

                profile.save()

                login(request, user,
backend='django.contrib.auth.backends.ModelBackend')

                #Rediects to index page.

                return redirect('/')

            else:

                return render(request,
'identyti_app/enterprise_sign_up.html', {'form': form, 'try_again': True})

                    #Create a USER that has an Enterprise object and the
attributes

    else:

        form  = EnterpriseSignUpForm()
```

The enterprise sign up view function contains the code that creates an enterprise account and associates it with the correct enterprise. This is done through verifying that the 4 digit code provided by the user attempting to sign up matches the hashed code stored in the database.

3.3.3 Forms

```
class DocumentForm(forms.ModelForm):

    class Meta:

        model = Document

        # <field in model> : <name you want displayed>

        labels = {

            "document": "Document",

            "type_of_document": "Type of Document"

        }

        fields = (labels)
```

The document form is used to create a document object in the database when a user uploads a document from the app. The Document entry has the information needed to retrieve the document from S3.

```
class EnterpriseSignUpForm(forms.Form):

    username = forms.CharField(widget=forms.TextInput(attrs={'placeholder':
'JaneDoe'}))

    enterprise =
forms.CharField(widget=forms.TextInput(attrs={'placeholder': 'My
Enterprise'}))

    email = forms.CharField(widget=forms.TextInput(attrs={'placeholder':
'Email'}))

    password = forms.CharField(required = True,
widget=forms.PasswordInput())

    confirm_Password = forms.CharField(required = True,
widget=forms.PasswordInput())

    code = forms.CharField(required = True,
```

```
widget=forms.TextInput(attrs={'placeholder': '0123'}))
```

This form is the one that is displayed when enterprise users are signing up. The information collected by this form is passed to the enterprise_sign_up view function to create a new enterprise account.

```
class EnterpriseLogin(forms.Form):
    username = forms.CharField(widget = forms.TextInput())
    password = forms.CharField(widget = forms.PasswordInput())
```

The enterprise login form is used to collect the username and password of enterprise users in order to authenticate them.

3.3.4 ML Model

Using sci-kit learn, the machine learning model was developed to be capable of categorizing documents that users upload. In Identyti, there are six main categories that a document can be classified as: Identification, Financial, Insurance, Medical, Work/Academic, and Legal.

```
class Classification:
    def __init__(self):
       ...
     def classify(self, file):
         try:
             docs = []

docs.append(pytesseract.image_to_string(Image.open(file)).strip())
             X_new_counts = self.count_vect.transform(docs)
```

```
            X_new_tfidf = self.tf_transformer.transform(X_new_counts)
            predicted = self.loaded_model.predict(X_new_tfidf)
            return self.doc_types[predicted[0]]
        except:
            return self.doc_types[1]
```

By leveraging Google's Tesseract OCR, which parses text information from images, the Classification class first extracts the text information from the document the user uploads, prepares the text information, runs it through the model, and returns the category. This simple process is run every time a user uploads a document, making the process more streamlined.

The model was built using hundreds of test documents unique to each category. The text information was extracted and input into the Naive Bayes classifier, which classifies based on discrete features between the data.

```
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(info)
tf_transformer = TfidfTransformer()
X_train_tfidf = tf_transformer.fit_transform(X_train_counts)
clf = MultinomialNB().fit(X_train_tfidf, categories)
```

This small snippet of the code highlights the main building blocks of the model. Using the given "info", which is the text information extracted from the test data, this data is transformed to prepare it for building the machine learning model. Finally, we combine this with "categories", which contains the labels for the given data (i.e. document 35 is an Identification document,

document 36 is a Financial document, ...) and fit this with the Naive Bayes classifier. This final "clf" is the classifier used in Identyti.

3.4 Sample Tests

Tests are essential to developing and maintaining any application. They help us ensure that we are implementing things correctly and they will also be important for the customer to verify that the app is still working as expected.

This test verifies that when two enterprises are created, they do not have the same secret code:

```
def testCodeStored(self):

        enterprise1 = Enterprise.objects.create(name="Test Inc",
email_contact="test@virginia.edu")

        enterprise2 = Enterprise.objects.create(name="Test2 Inc",
email_contact="test@virginia.edu")

        assert enterprise1.code is not None

        assert enterprise2.code is not None

        assert enterprise1.code != enterprise2.code
```

This test uploads a file to S3 and saves it as an entry in the test database, then verifies that the file path was saved correctly.

```
def test_document_filepath_ID(self):

     num = random.random()

     form_data = {

         'type_of_document': "ID",
```

```
    }

    form_files = {

        'document': SimpleUploadedFile("test{0}.png".format(num), b'content
of the file'),

    }

    form = DocumentForm(data=form_data, files=form_files)

    document = form.save(commit=False)

    document.user = User.objects.get(pk=1)

    document.owner = Profile.objects.get(pk=1)

    document.save()

    assert str(document) == "testuser/ID/test{0}.png".format(num)
```

3.5 Code Coverage

We used coverage.py to measure our code coverage. To install it we ran ‘pip install coverage’ and ran ‘coverage run manage.py test’ to run the tests and ‘coverage html’ to create an html report.

Our product reached 99% code coverage, only missing a few lines that are untestable due to their random/secure nature or the fact they only are used on the heroku deployment.

3.6 Installation Instructions

Our source code can be found here:

https://drive.google.com/file/d/1fAAG--VXZQa3cmP4ZXUsMYIU8vhFrAiu/view?usp=sharing

This code uses Django, social-auth-app-django, boto3, django-storages, gunicorn, django-heroku, and django-crispy-forms. These will automatically be installed by Heroku.

3.6.1 Setting up the Repo:

Skip this section entirely if you've been provided the code base within a non-organization Github repo. You can also skip to the Create Repo and Upload section if you already have a Github account.

Git and Github are important tools in software development. They allow for version control and give an integrated system to back up code and restore it in case of computer failures. As such, it's good practice to store code in a github repo. This will also make CI/CD(Constant integration and constant deployment) possible. **This section outlines how to take a zipped folder and upload it to github.**

Begin by creating an account on https://github.com/ . Enter your desired username, email, and password and solve the puzzle to confirm you are not a robot. Your password must be either at least 15 characters long, or 8 characters with a number, an uppercase, and a lowercase. Once this is filled in click the "Select Plan" button. This will move you forward to a plan selection page. Select the 'Individual' and 'Choose Free' options:

On the next page select "skip this step." Check your email for your github account verification email and follow the instructions in the email. Sign in and go to the github Homepage by clicking on the cat-like icon in the top left corner.

Create Repo and Upload: ***start here if you already have a Github account***

On the home page select the “Start Project” button:

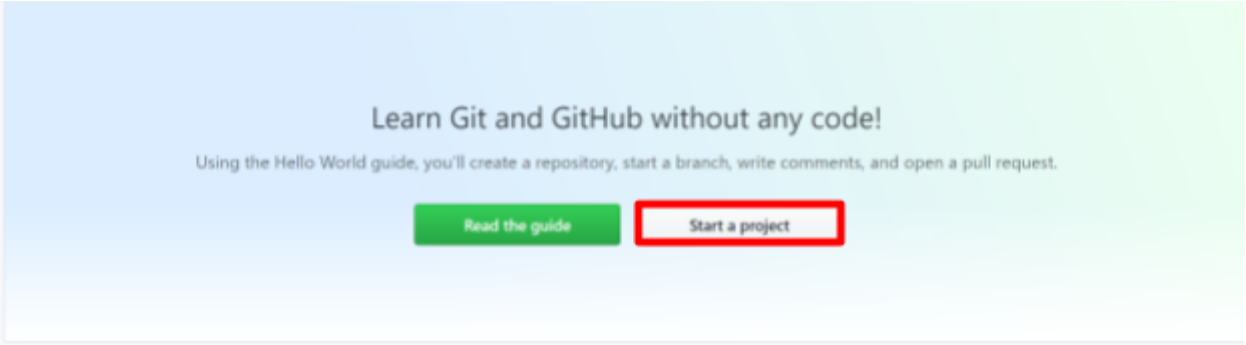


Figure 3

You should now be seeing the form to create a new repository. Enter in the name of the project(ie “Identyti”) and a brief description. Select the “Private” option. Leave “Initialize this repository with a README” unchecked. Click create.

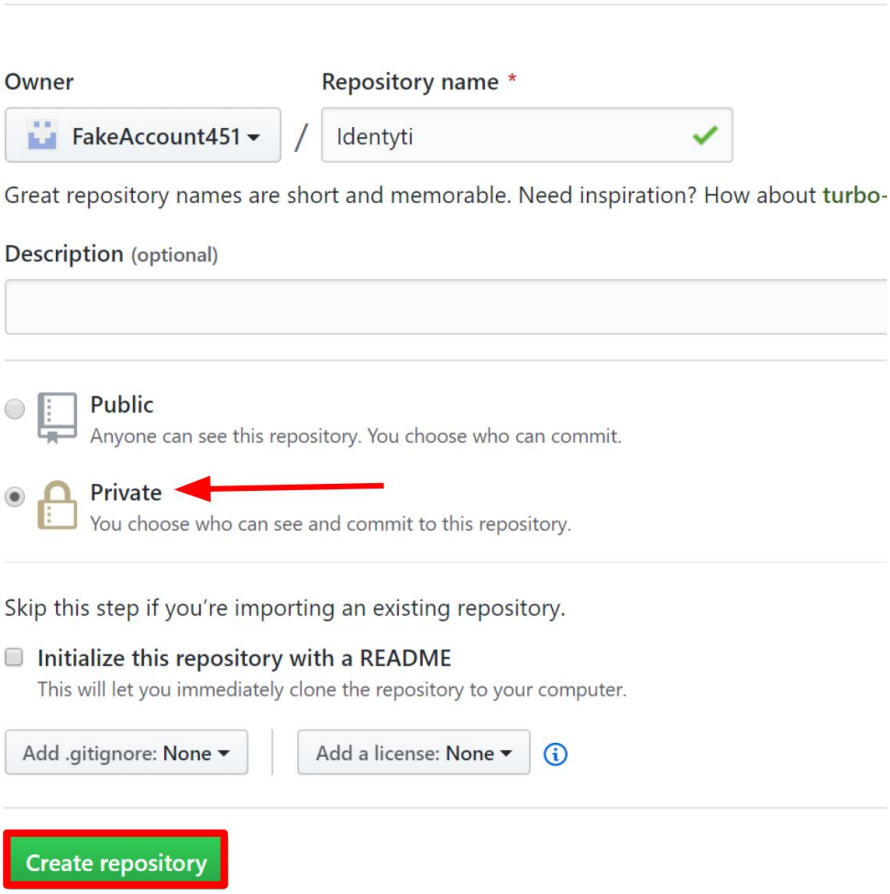


Figure 4

You now have a repository on Github! Congratulations!

Now on your own computer, find your zip folder and extract its contents to a location you can find. The steps for this will vary based on your operating system.

Now choose one of the following two methods to upload your extracted files to the Github Repository.

Github CLI(Linux): ***skip if using Windows or Mac and you are not familiar with the command line***

Use apt-get or a similar package manager to install git. Clone the repo and copy and paste the project files into the folder created. Git add and commit those changes back to the repo.

Github Desktop(Windows, Mac): ***THIS IS THE PREFERRED OPTION FOR THOSE WITH NO PRIOR TECHNICAL EXPERIENCE***

We are now going to install Github Desktop. This is a tool which allows you to interact with github from your desktop. **If you already have, and are familiar with Github Desktop, Git CLI, or any tool that lets you interact with Github remotely, feel free to skip these steps and use those tools.**

Navigate to https://desktop.github.com/ and click on the download link for your OS. Run the downloaded executable file. If prompted click “Install Anyway” on Windows.

If prompted to sign in, sign in to the github account you just made. If not, sign in by going to the “File” dropdown, and selecting “Options”. In the window that opens, click account(if necessary) and enter your information to sign in. On the page that opens select the repo we just created and click clone.

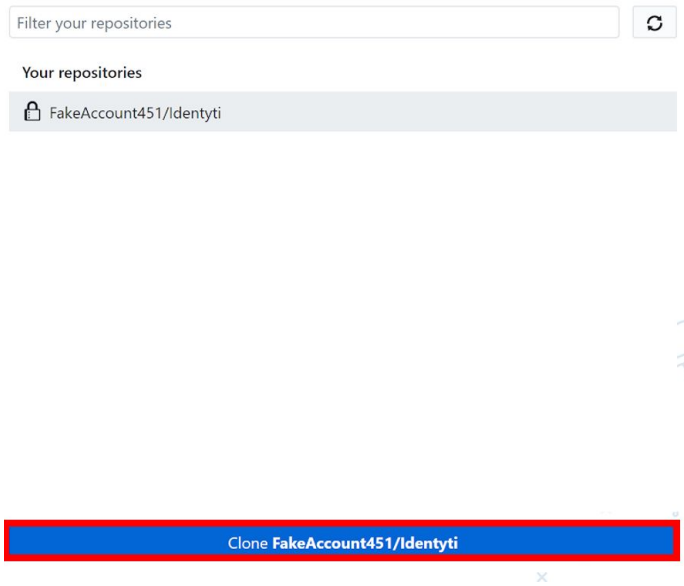


Figure 5

Click show your files of your repository in Explorer. This will open a file explorer window that shows your repo files. There should not be any files in the repo yet.

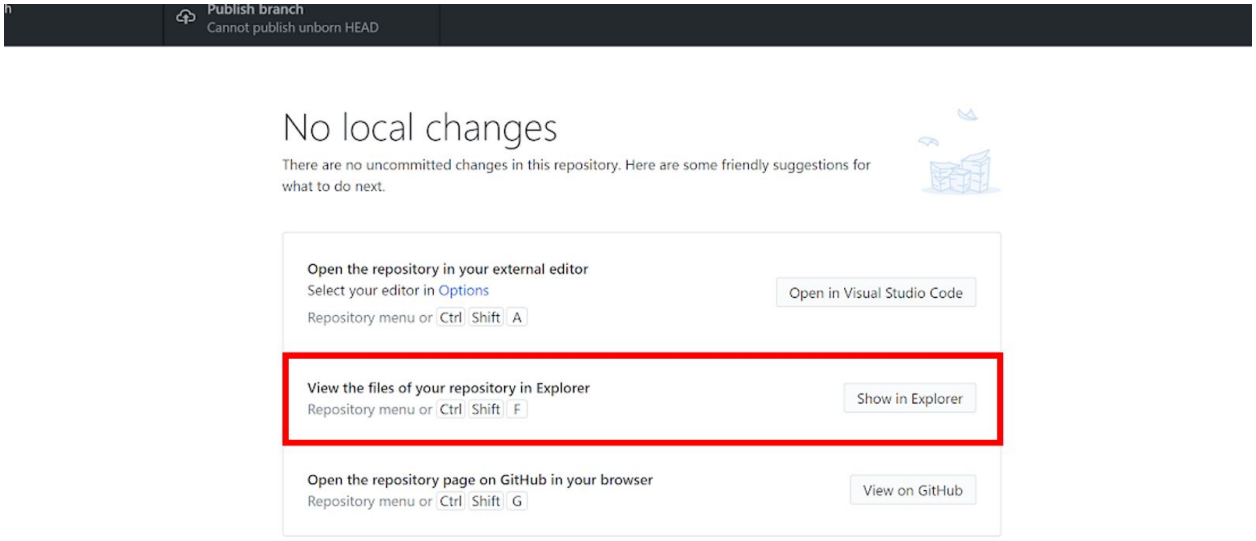


Figure 6

In a separate file explorer window, navigate into the folder we extracted earlier. Click into the folders until you see a list of files such as the following:

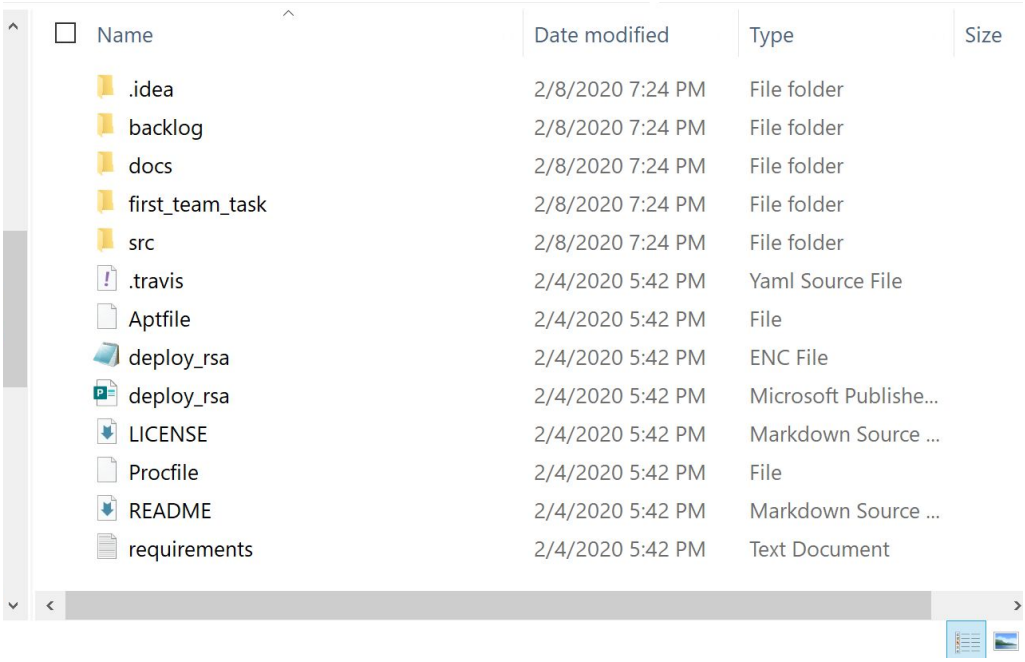


Figure 7

If you don't see a "src" folder, you're in the wrong section. My file path is along the lines of Desktop/identyti_master/identyti_master/.

Copy all the files from this folder into the empty folder opened by Github Desktop. You can do this in windows by clicking the explorer window with files, hitting ctrl+A, ctrl+c, switching explorers, and hitting ctrl+v.

Return to github desktop. You should now see a long list of changes on the left like this:

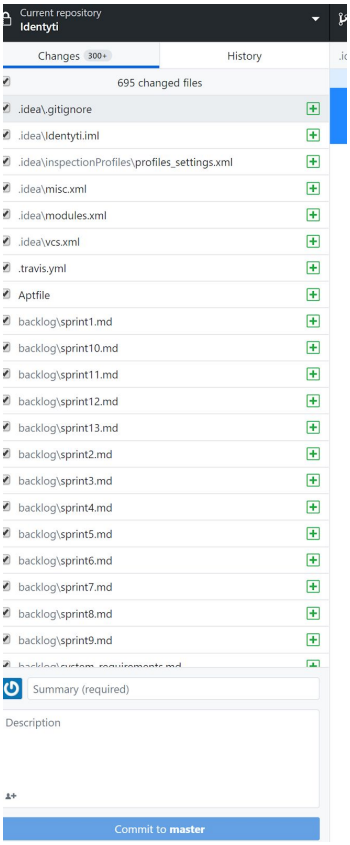


Figure 8

At the bottom of that list, enter into the summary box and type “First Commit.” Click “Commit to master.”

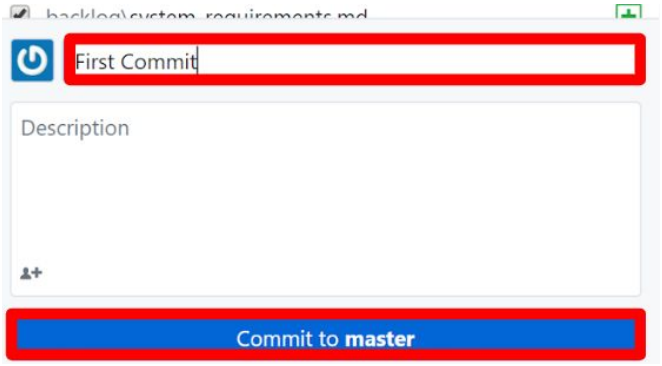


Figure 9

Click “Publish Branch”

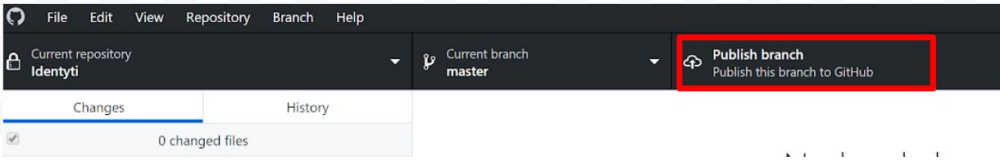


Figure 10

You now have successfully created a repo with all the files we need for deployment.

3.6.2 Deploying with Heroku:

Navigate to https://www.heroku.com/ and sign up for an account.

Click Create a new app on the homepage

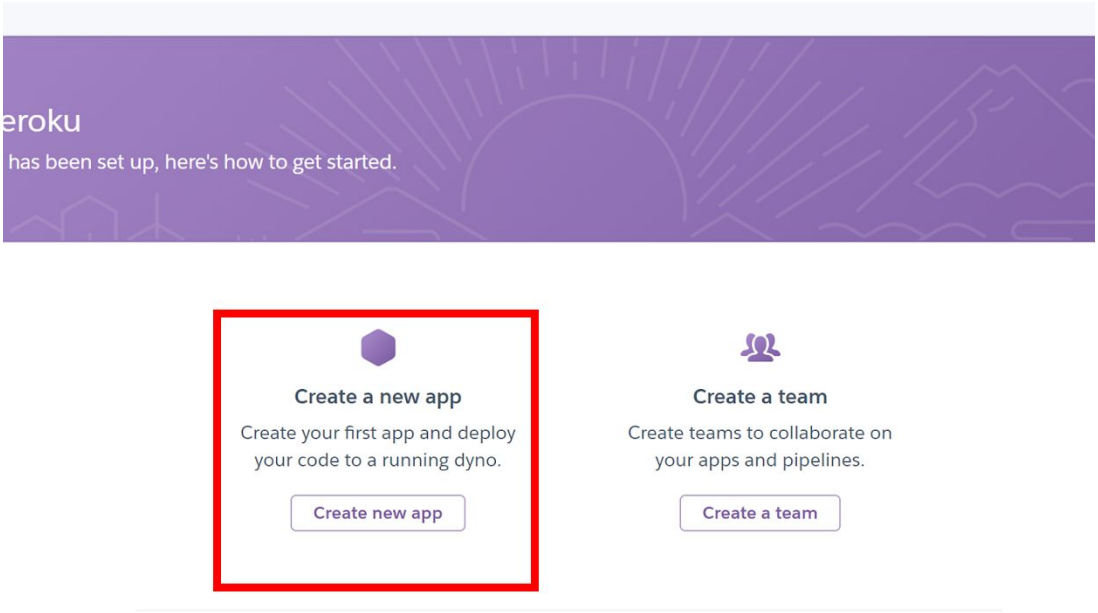


Figure 11

Choose a unique app name and place it into the name box. Click create app

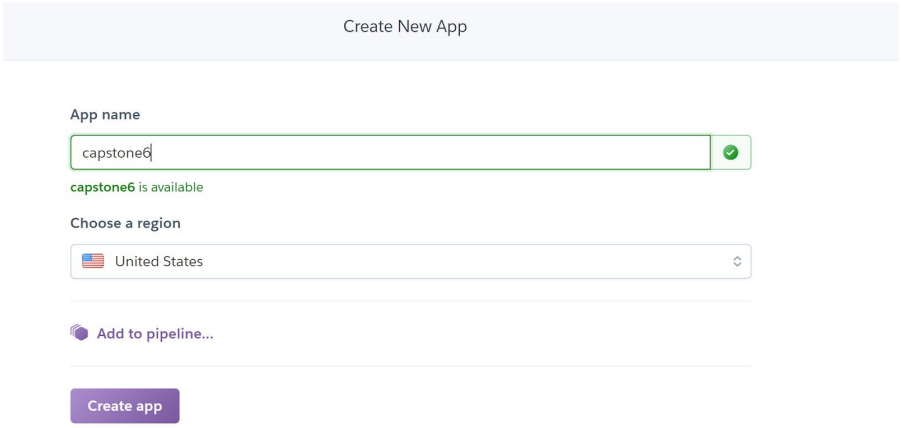


Figure 12

Heroku should navigate you to your app's dashboard. If not already there, click "Deploy" at the top. Next to "Deployment method", click "Connect to Github". In the section that opens, click "Connect to GitHub".

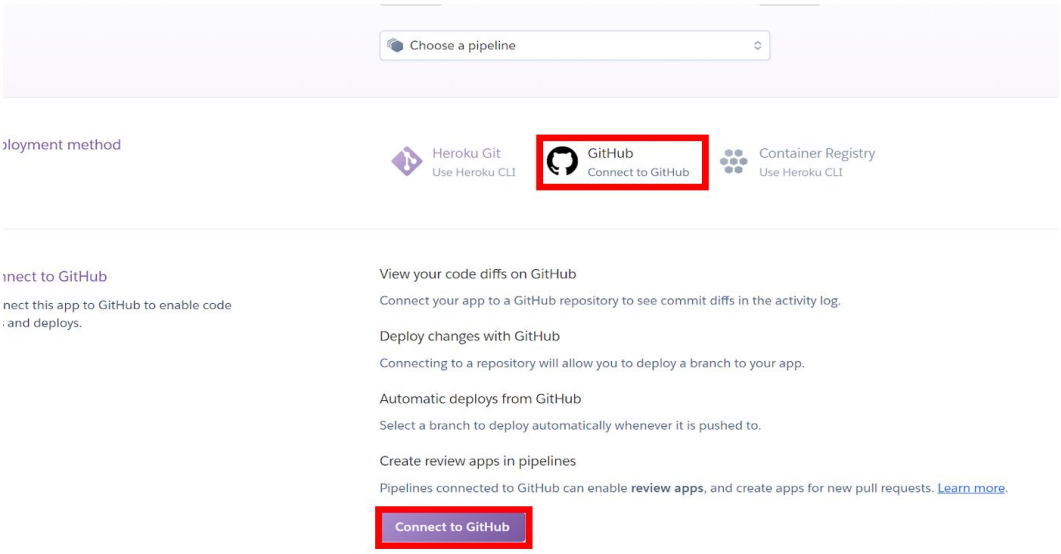


Figure 13

This should open a prompt that asks you to authorize Heroku to access your GitHub. If this window does not open your browser may be blocking pop-ups. Click Authorize in this new window.

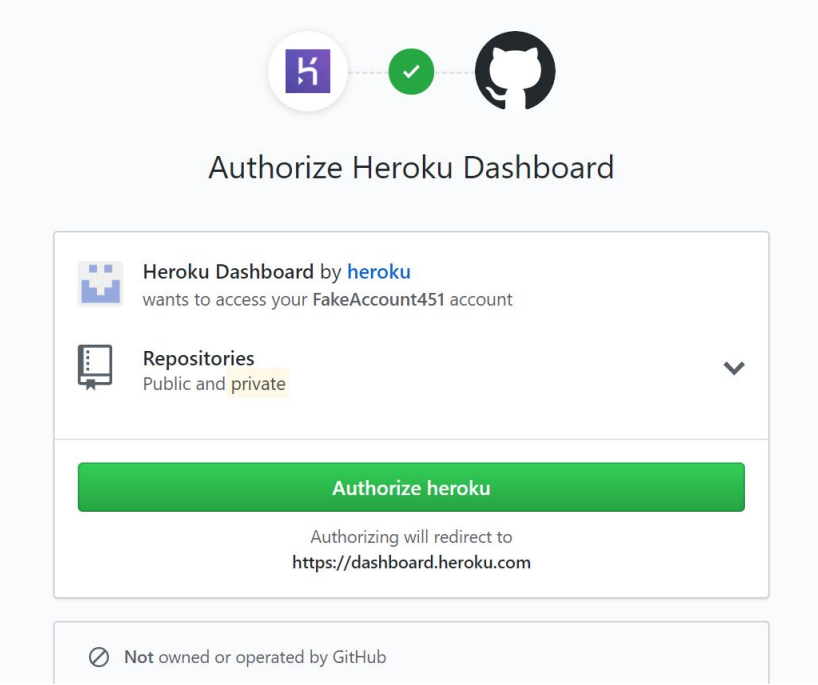


Figure 14

Heroku should now show you a search bar. Enter your repo name in that search bar and search for your. Click “Connect” next to your repo when it shows up

Click connect

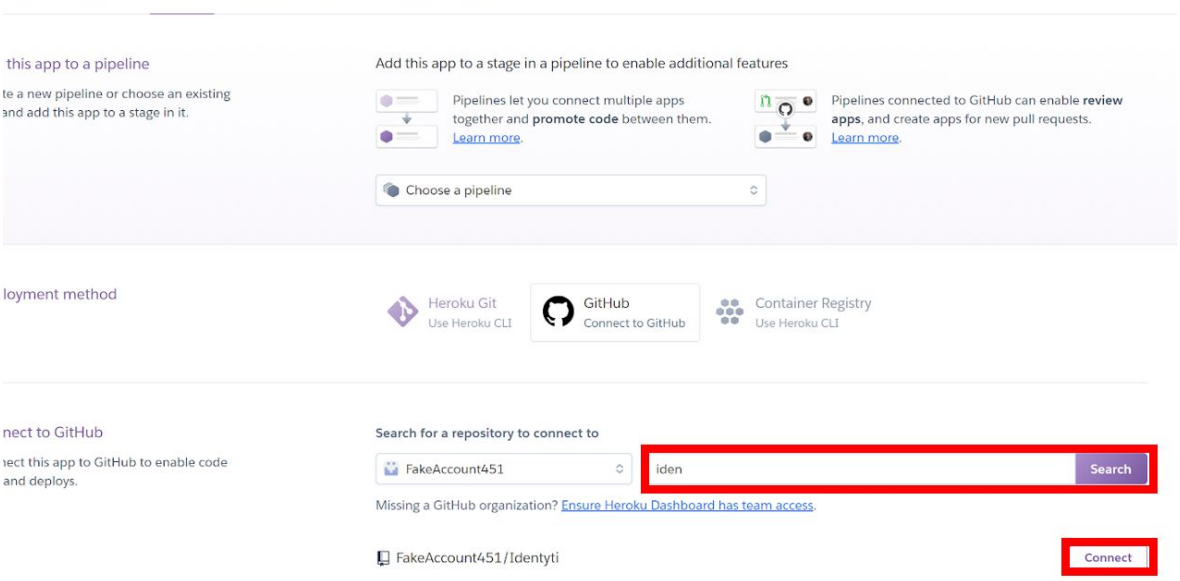


Figure 15

Click the box “Enable Automatic Deploys”

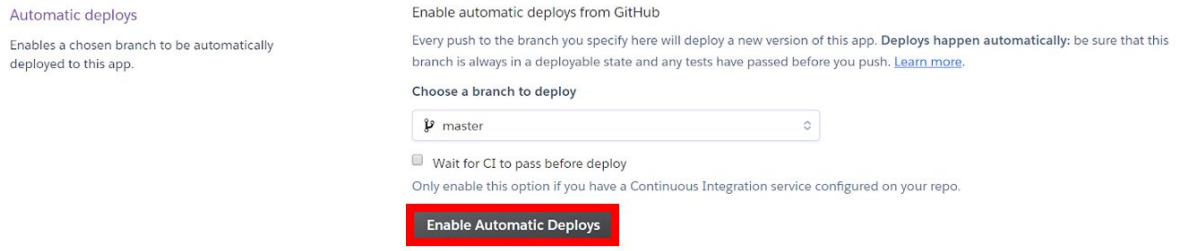


Figure 16

Next to the Manual deploy section click “Deploy Branch”

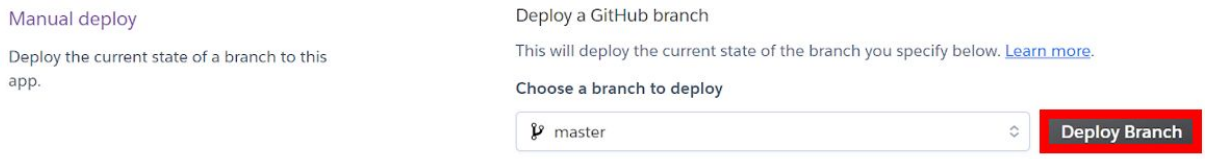


Figure 17

After a few minutes, you should get the message that your app was successfully deployed. Click View! You should see a working app.

3.6.3 Configure the Database:

Next we need to set our database. Heroku provides a free Postgres Database for projects to use, which is what we'll use. On your Application's Dashboard click "Overview" and then "Heroku Postgres"

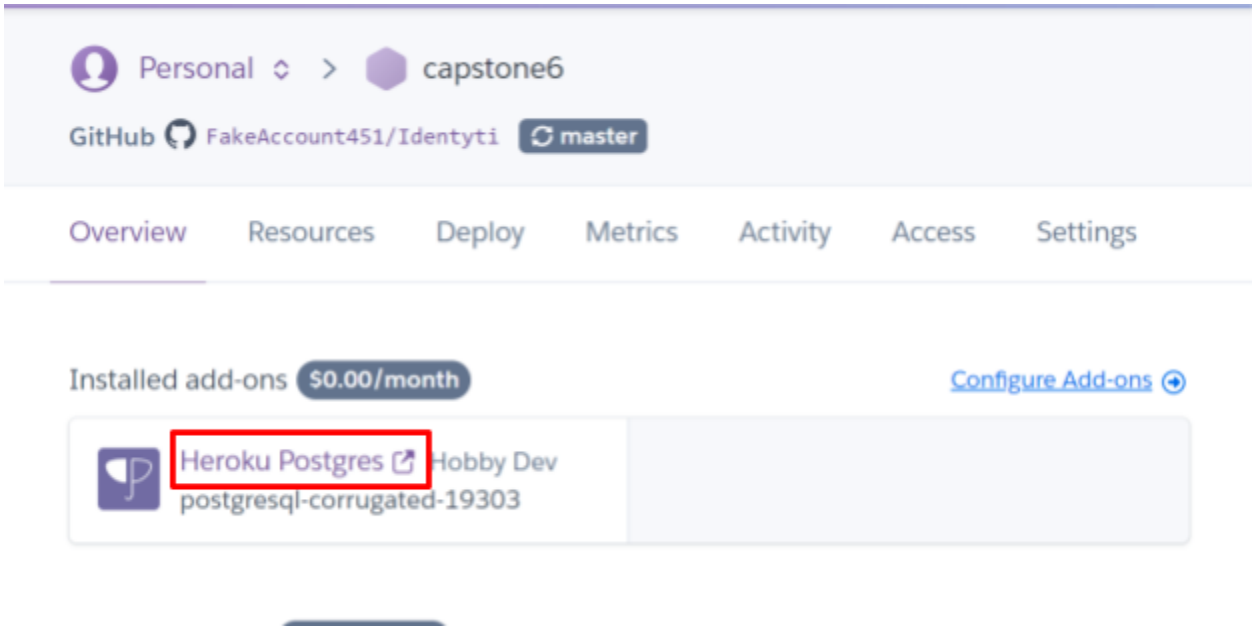


Figure 18

On the new page click “Settings”

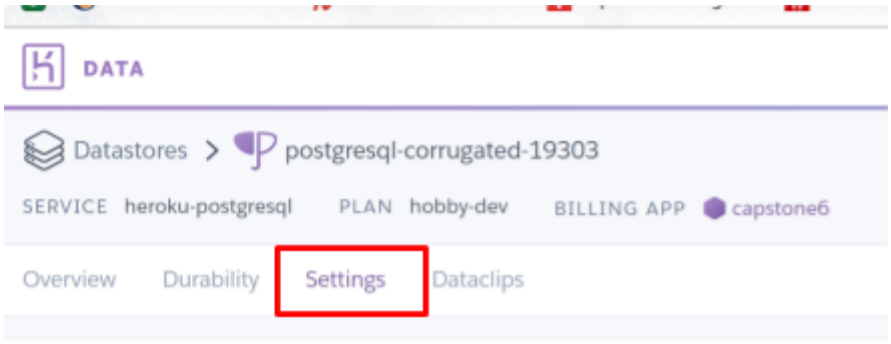


Figure 19

Next to Database Credentials click “View Credentials”

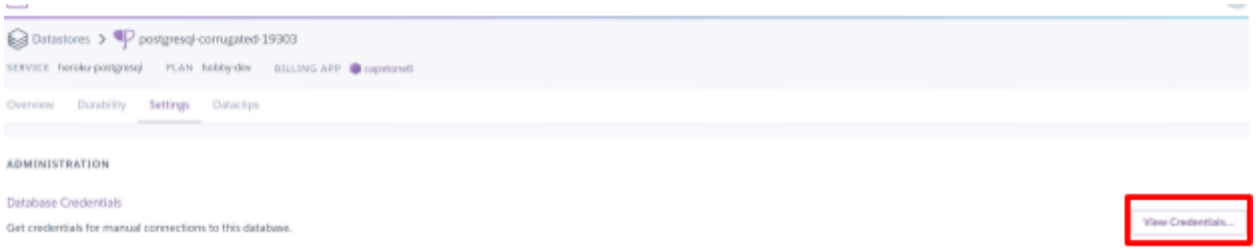
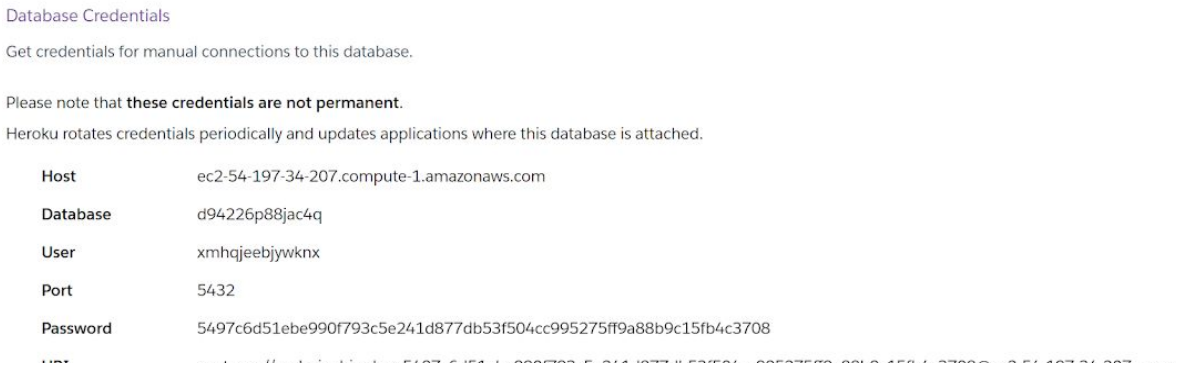


Figure 20

You should see something like this:



Open Github in a new tab and navigate to your repo. Inside the repo go to the “src/identyti” folder, and then “identyti” folder, then “settings.py”. Hit the small pencil icon on the top right of the file to edit the file.

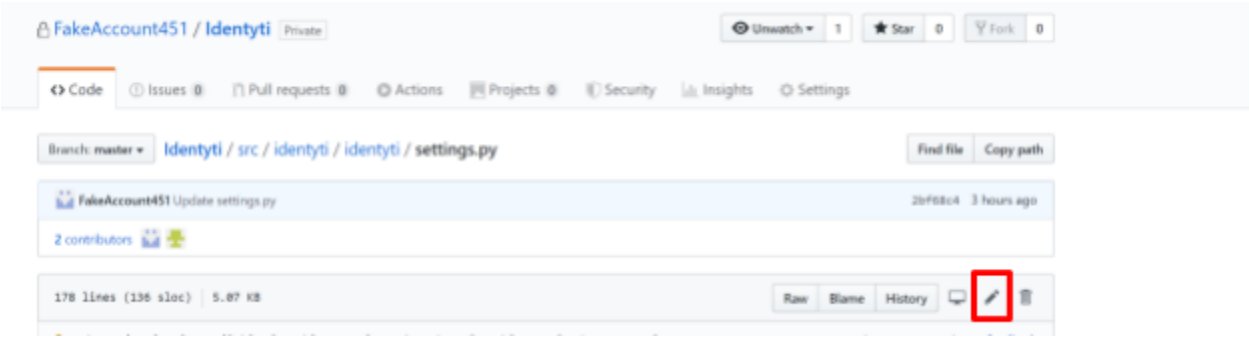


Figure 21

Scroll to the “Database” section of the code. If you want, you can use your browser’s find function to search, or you may look yourself. It looks like this:

```
80  if 'HEROKU' in os.environ:
81      DATABASES = {
82      'default': {
83          'ENGINE': 'django.db.backends.postgresql_psycopg2',
84          'NAME': 'd35rolpv2jjqpg',
85          'USER': 'zsdpylesfkfkre',
86          'PASSWORD': '95e2ee896550c76aae8ad549e2be89cb54dda5e48c487787a2574df9bfa1722e',
87          'HOST': 'ec2-174-129-253-157.compute-1.amazonaws.com',
88          'PORT': '5432',
89      }
90  }
91  else:
92      DATABASES = {
93          'default': {
94              'ENGINE': 'django.db.backends.sqlite3',
95              'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
96          }
97      }
```

Figure 22

Now, we’re going to copy and paste values from our Heroku credentials into this file. Replace the NAME, USER, PASSWORD, and HOST sections under the if ‘HEROKU’ line. Name will

be called “Database” in your credentials. It is important to place these values within a set of single quotation marks when you change them.

```
80  if 'HEROKU' in os.environ:
81      DATABASES = {
82      'default': {
83          'ENGINE': 'django.db.backends.postgresql_psycopg2',
84          'NAME': 'd35rolpv2jjqpg',
85          'USER': 'zsdpylesfkfkre',
86          'PASSWORD': '95e2ee896550c76aae8ad549e2be89cb54dda5e48c487787a2574df9bfa1722e',
87          'HOST': 'ec2-174-129-253-157.compute-1.amazonaws.com',
88          'PORT': '5432',
89      }
90  }
91  else:
92      DATABASES = {
93          'default': {
94              'ENGINE': 'django.db.backends.sqlite3',
95              'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
96          }
97      }
```

Figure 23

Hit “commit changes” at the bottom of the file

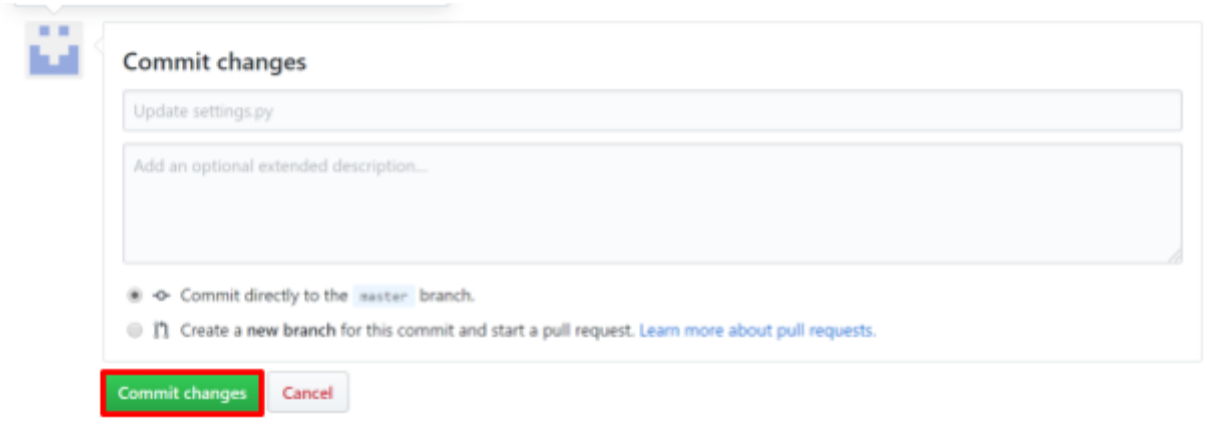


Figure 24

We've now configured our database to use Heroku's provided Postgres Database.

3.6.4 Configure OAuth:

We use Google OAuth to sign in most users for this application. We'll have to make some small changes to the code to configure OAuth for this new deployment.

Begin by going to https://console.cloud.google.com/ and signing in with google.

Confirm your country and agree to the Terms of Service.

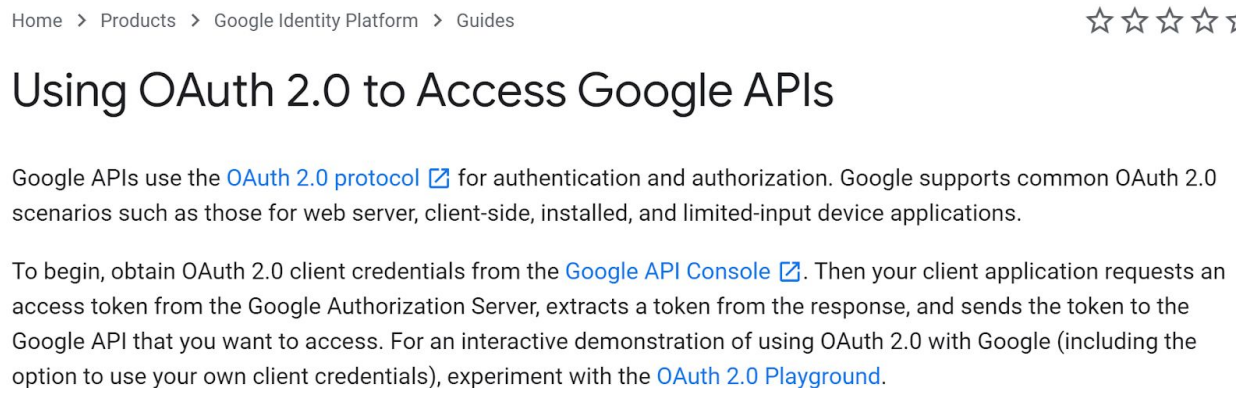


Figure 25

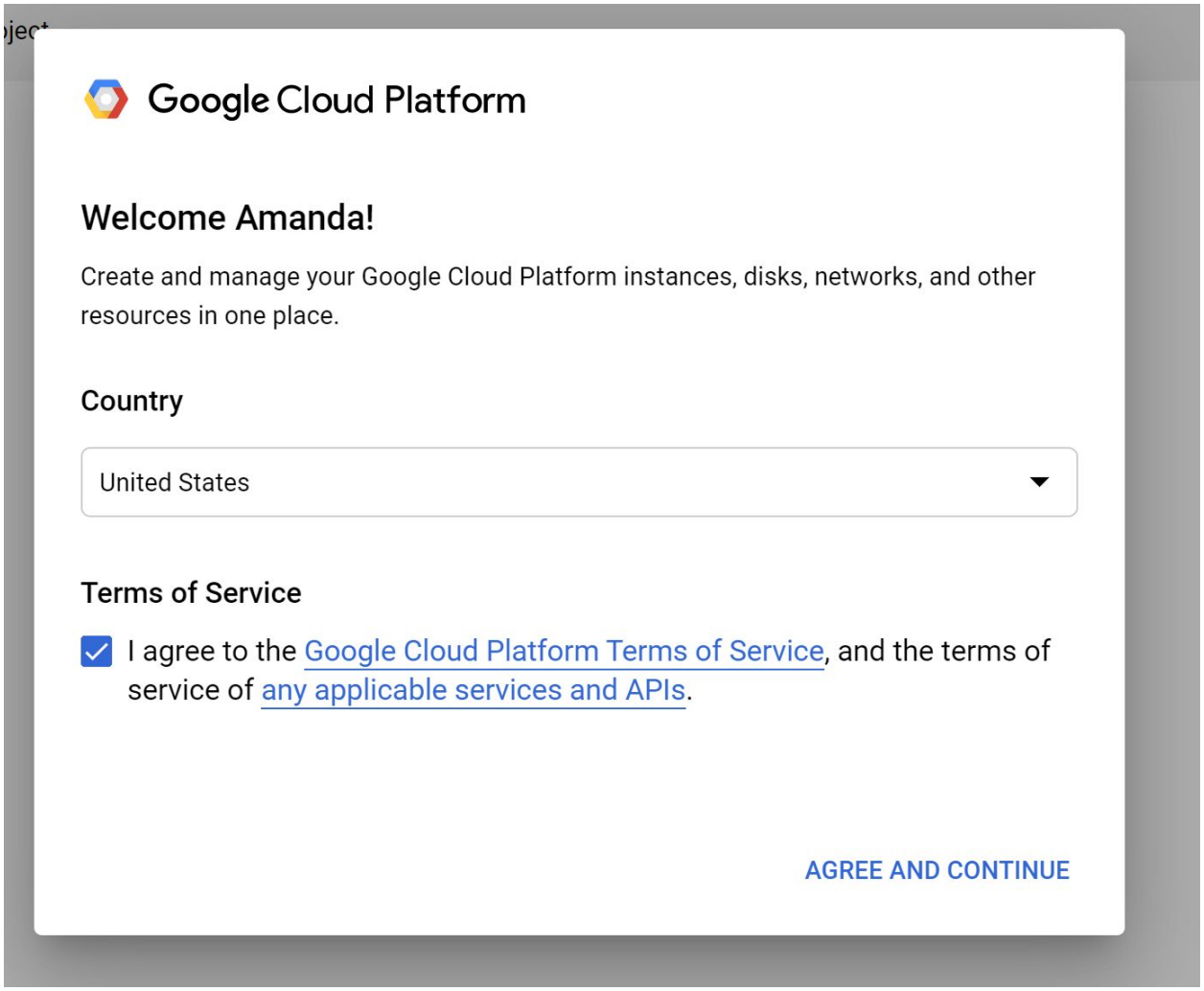


Figure 26

Click Credentials -> Create . This will open a dialogue that will allow you to create a project that uses Google's Services.

Figure 27

Enter a project name and click create.

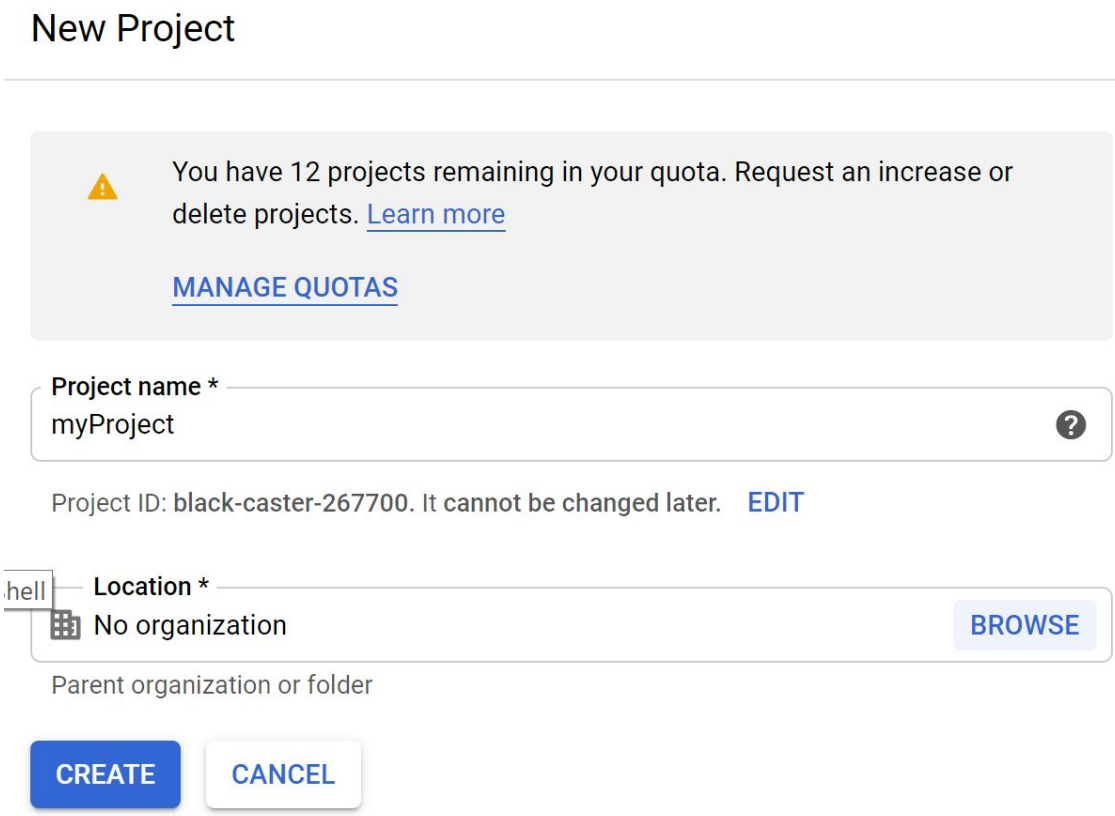


Figure 28

Next select the menu button in the top left corner, select API's & Services, then Credentials.

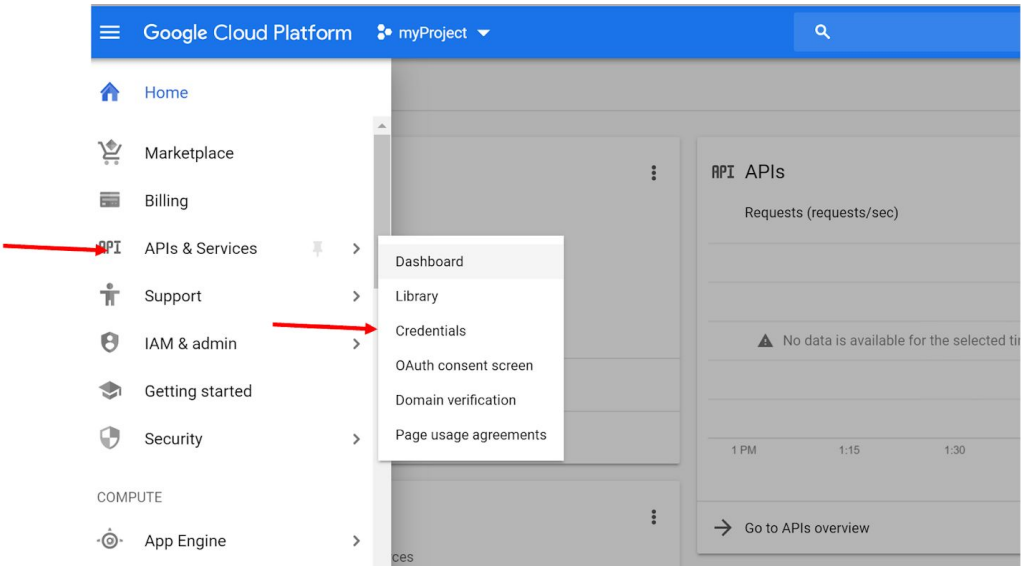


Figure 29

First you will need to confirm ownership of your domain, click domain verification on the left, enter your domain name and click Add Domain. Copy your heroku domain into the box. It will ask you if you would like to verify, choose 'take me there.' This will open a new tab.

On the new page, click "Add a Property" and then "Alternate methods". Choose the "HTML tag" option. This will open a box that has a tag that looks like this:

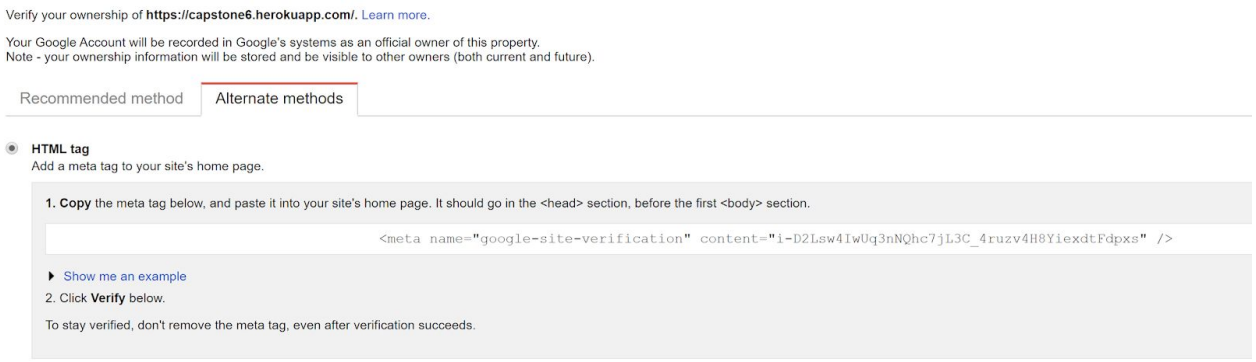


Figure 30

In a separate tab, open Github and then navigate into the "src/identyti" folder, than "templates/identyti_app", and finally "base.html" and click the pencil to edit. There should already be a line there that looks like this

```
-->
<!-- Google Auth -->
<meta name="google-site-verification" content="Y12J7ZCfapFtXQy3tXZQ7iRMBkUs3-JlYhz2sHqo4DI" />
```

Figure 31

Replace that line with your new one, making sure to keep the indentation the same. Click "Commit changes" at the bottom. Wait for a few minutes as Heroku creates and deploys a new build. You can check the logs on your heroku dashboard to see when these changes are deployed.

Return to the "Webmaster Central" tab and hit "Verify." You should see a screen like this:

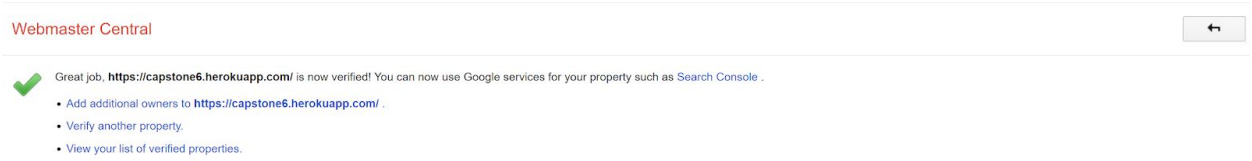


Figure 32

You can close out of this tab now, and return to your Domain Verification tab.

Chose OAuth consent screen from the left menu. Choose external and then fill in your application name and homepage and click save.

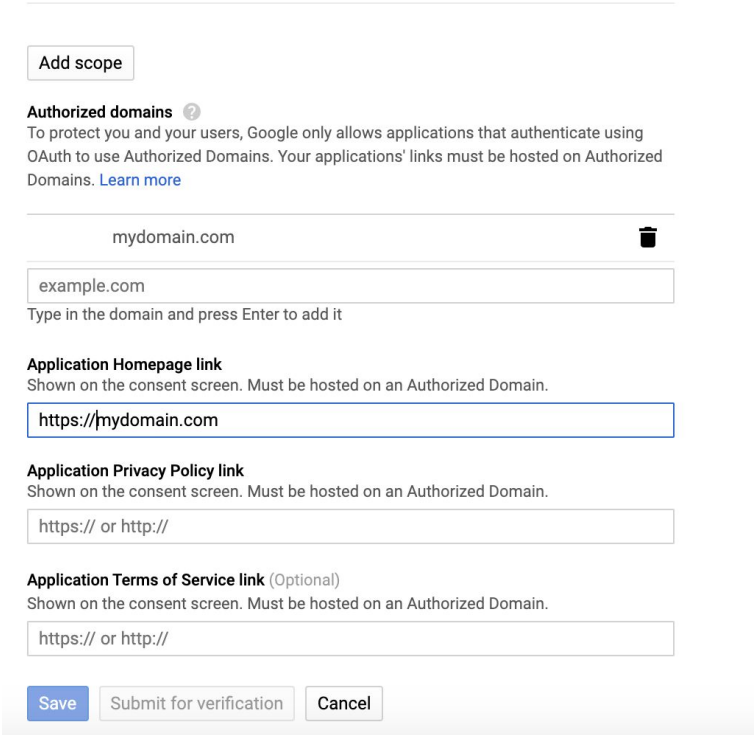


Figure 33

Next click on the credentials tab, click + CREATE CREDENTIALS at the top and select OAuth Client ID. On the next page select Web Application and in the Authorized URI's section add your domain (using https://) and in the authorized redirect URIs section add your domain with /google-oauth2/ on the end of it and /complete/google-oauth2/. It should look like the ones below except you will not need localhost on yours

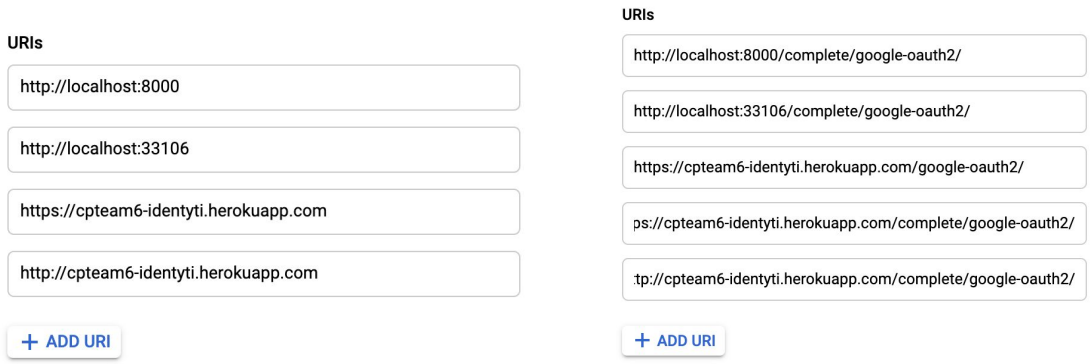


Figure 34

Now click the Create button at the bottom of the page. You will get a pop up with a set of credentials. Leave that open.

Now in your Github tab, find the settings.py file again, and click to edit. Scroll until you find the lines with "SOCIAL_AUTH_GOOGLE_OAUTH2_KEY" and "SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET".

```
LOGIN_URL = '/auth/login/google-oauth2/'
LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/'
SOCIAL_AUTH_URL_NAMESPACE = 'social'
AUTH_PROFILE_MODULE = 'identyti_app.Profile'


SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = '284603850335-fpcvmpr0uo2l869h7o81ho60sdlams51.apps.googleusercontent.com'
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = 'kR097Al1NlBBxMkXhbHJ8WAS'

django_heroku.settings(locals(), test_runner=False, staticfiles=False)
```

Figure 35

Change those values to match the credentials provided by Google. SOCIAL_AUTH_GOOGLE_OAUTH2_KEY will hold the value Google calls “Your Client ID” and SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET will hold “Your Client Secret”. Be careful place these values inside a set of single quotes. Once you’ve made your changes, click “Commit changes.”

Once Heroku deploys these changes you should be able to log in to your application with Google.

3.6.5 Setting up AWS

We use AWS to host user’s sensitive files. We made this decision because AWS has tried-and-true security solutions, and are a safe bet to keeping private data private. As such, you’ll have to configure an S3 bucket to hold important data in and an IAM account to upload that data. We’ll do that here.

Start by going to https://aws.amazon.com/ . Click “Create an AWS Account” in top the right.

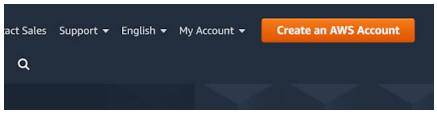


Figure 36

Follow the steps to create an account. You will have to provide a credit card-- AWS should not charge you beyond a basic "test charge" that will be quickly refunded.

When prompted choose the "Basic" plan.

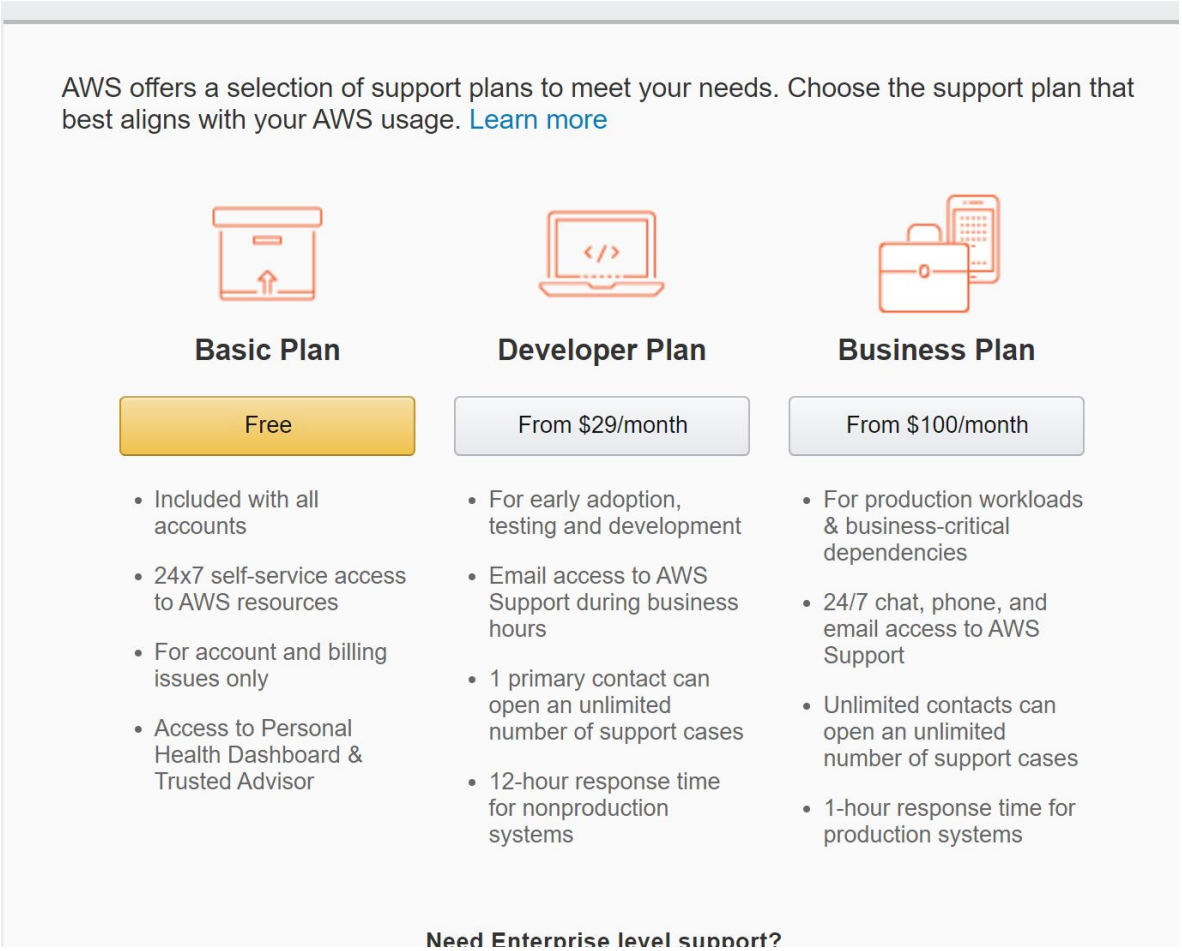


Figure 37

This gives you access to AWS's free tier. AWS gives you 12 months of free tier usage, which should work for early stages of the product. After that, you'll need to look into using paid options.

JUMP HERE IF YOU ALREADY HAVE AN AWS ACCOUNT

Click "Sign into the Console". Follow the steps to sign in. This will take you to your AWS console dashboard. Click "Services"

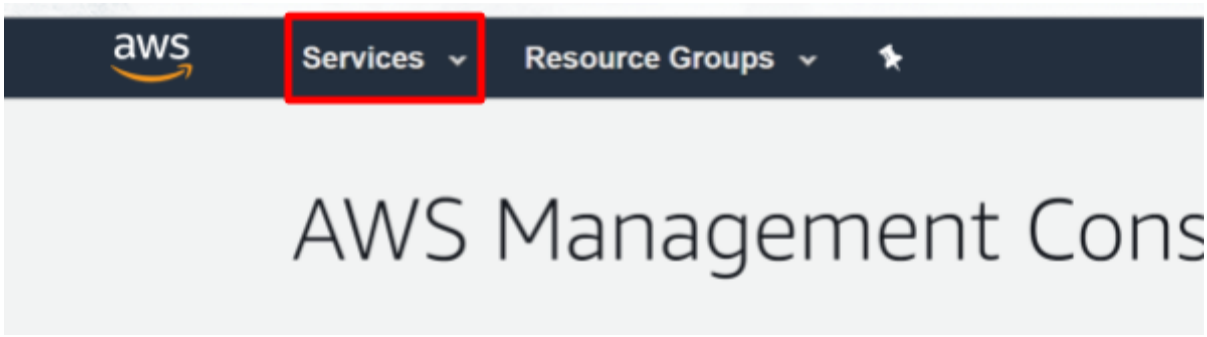


Figure 38

This will open up a *huge* selection of services. We only care about s3 right now. Click the S3 button under “Storage”

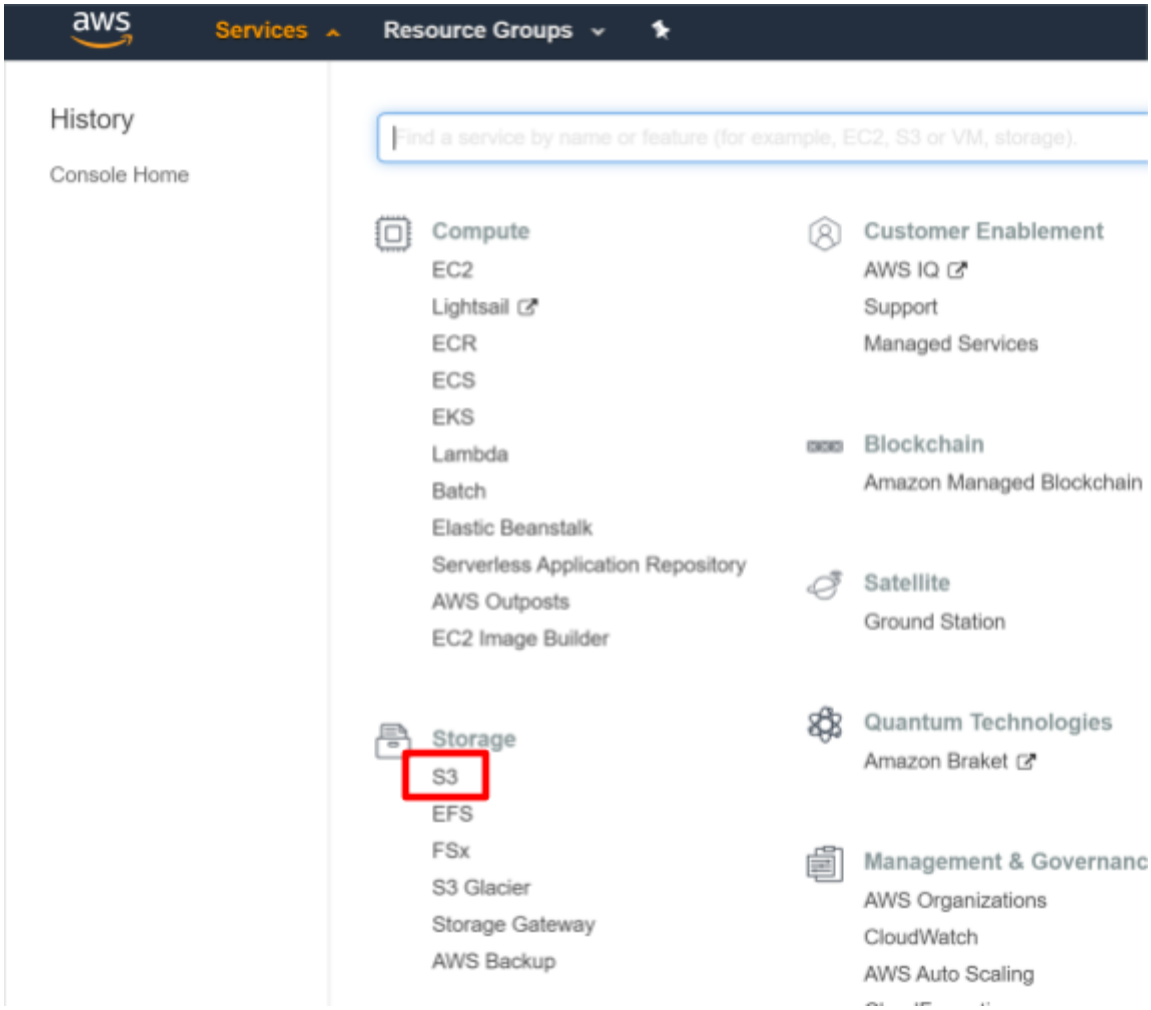


Figure 39

Click “Create bucket”

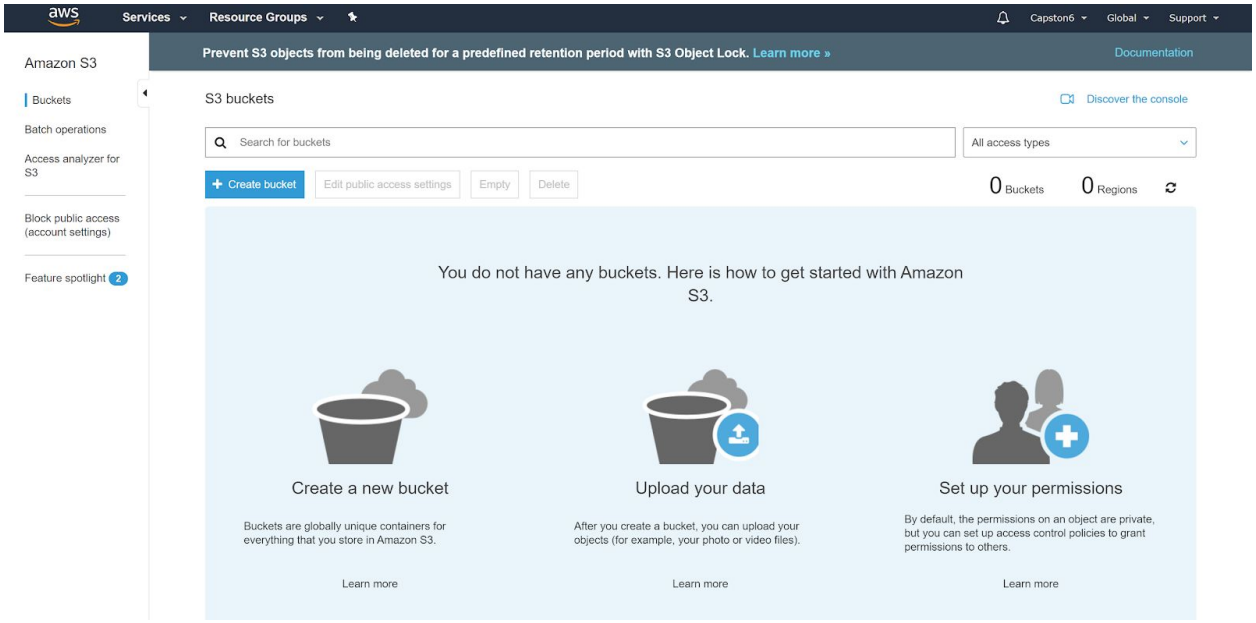


Figure 40

Enter a name for this bucket. AWS requires all bucket names to be unique across a region, so you may have to try a few names. Once you have a name, **choose US-east-1(US East N. Virginia) as your region. THE CODE IS NOT GUARANTEED TO WORK ON OTHER REGIONS.** Click "Next". Click "Next" again without selecting anything on the Configure Options page. The first box on the "Set Permissions" page should be checked. If it is not, check it. Make sure the drop down at the bottom is marked "Do not grant Amazon S3 Log Delivery group write access to this bucket". Click "Next". Your review should look something like this:

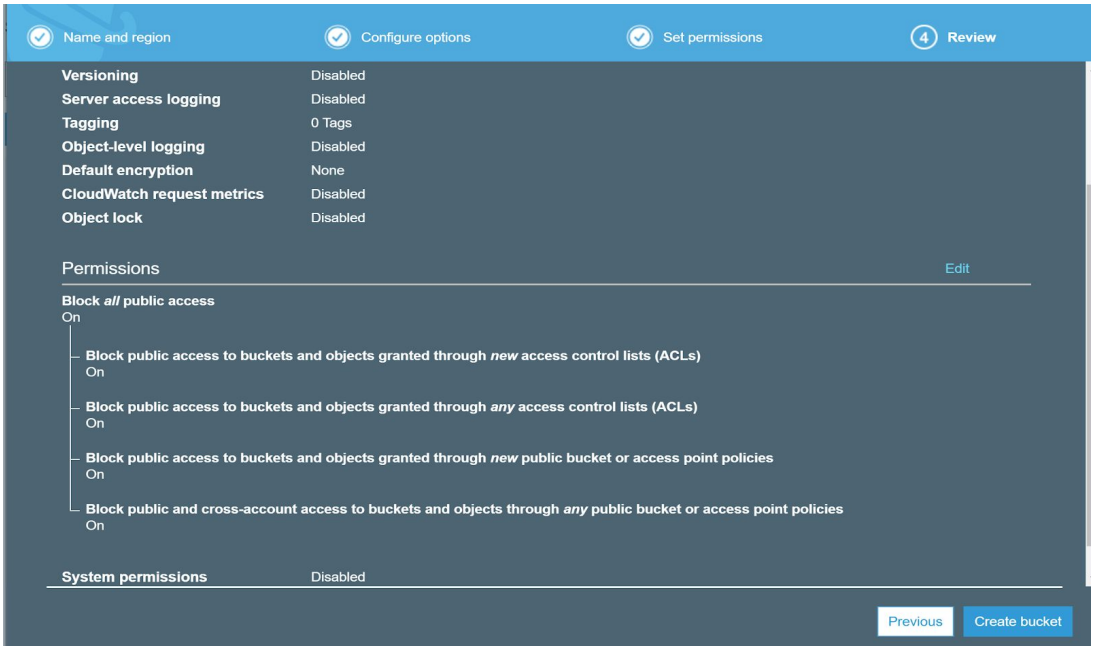


Figure 41

Click create bucket.

Inside the bucket click “Create Folder” and create a folder titled “media”. Don’t worry about encryption(stick with “None(use Bucket Settings)”).

Next we’re going to create an IAM user so we can add and remove things from this bucket.

Click the Services button in the top left again, search for “IAM”

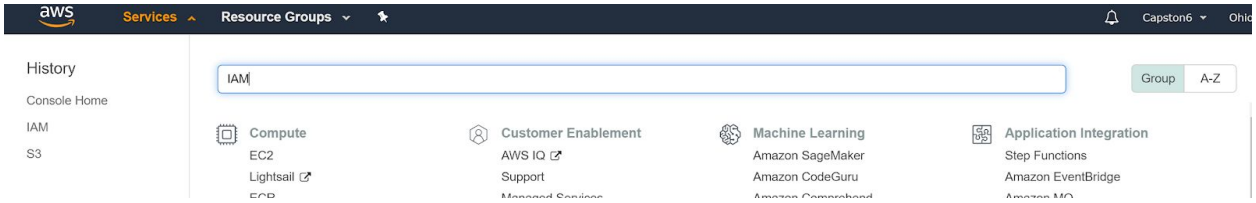


Figure 42

Click “Users” on the left hand side. Then click “Add User”

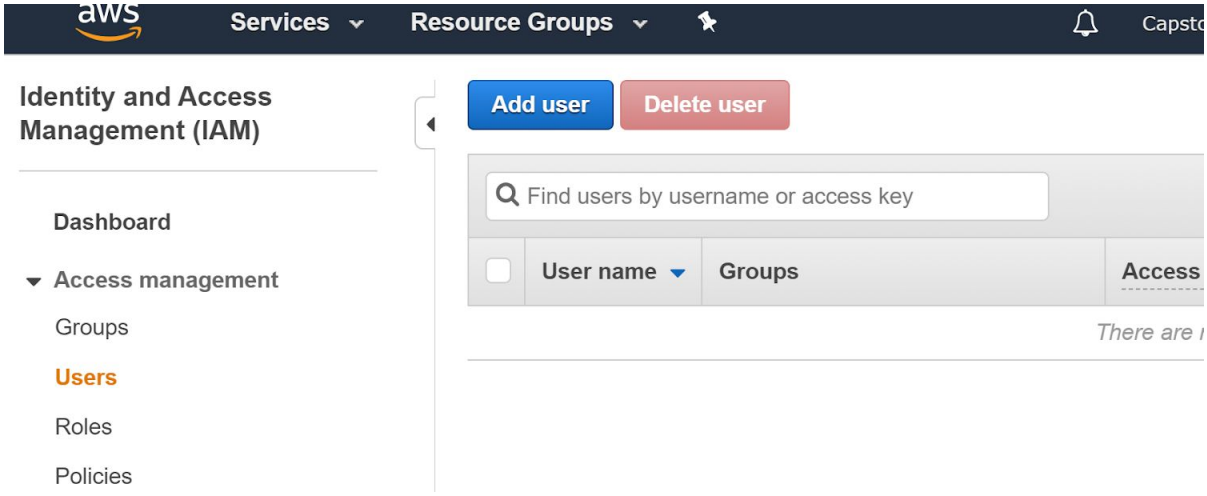


Figure 43

Choose a username for this user, and check the box next to “Programmatic access.” Click “Next: Permissions”.

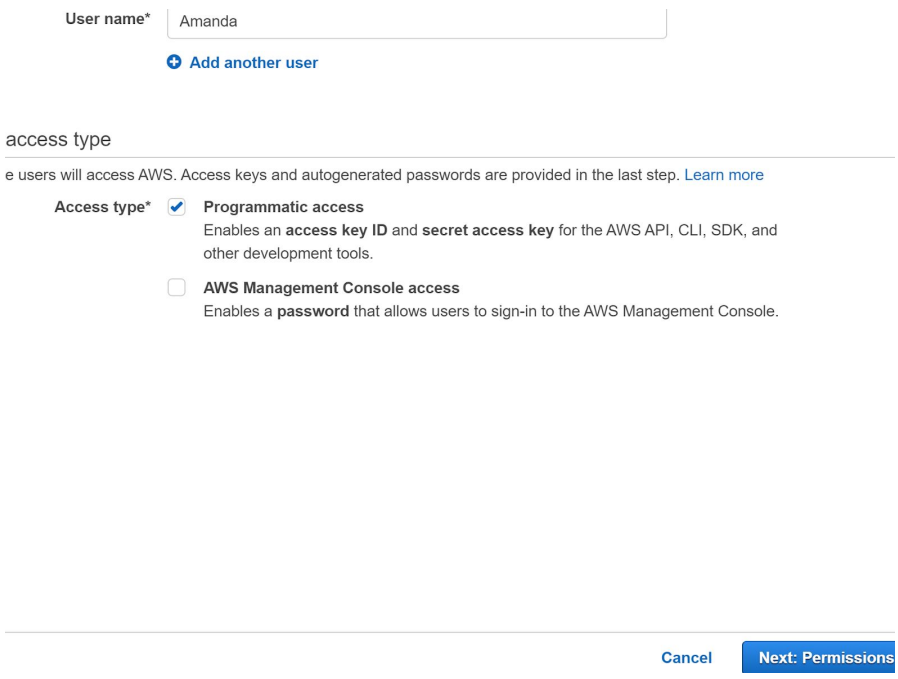


Figure 44

We will now need to create a new user group and add our user to it. Do this by clicking "Add User to Group" on the next page, and then clicking "Create Group".

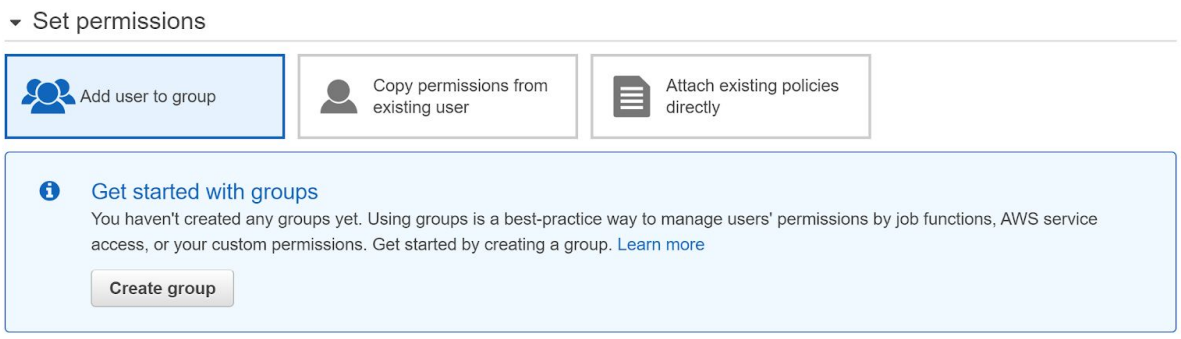


Figure 45

Choose a name and then search “AmazonS3FullAccess” in the search box. Select the policy that comes up and click “Create group”.

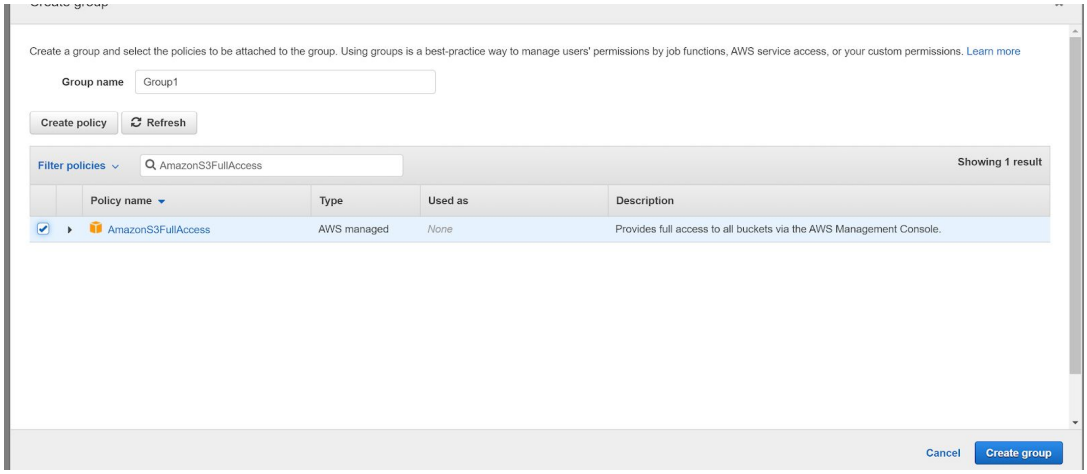


Figure 46

Click “Next: Tags” and “Next: Review”. Your Review should look similar to this:

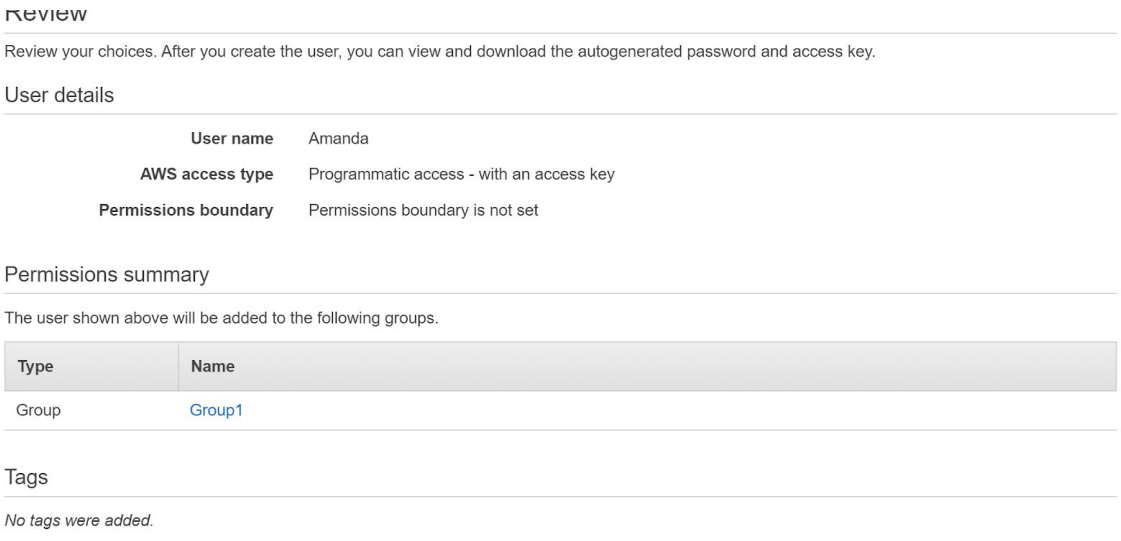


Figure 47

If everything is correct, click Create User. This will take you to a new page which has information such as your new User's Access key ID and Secret access key. It should look like this:

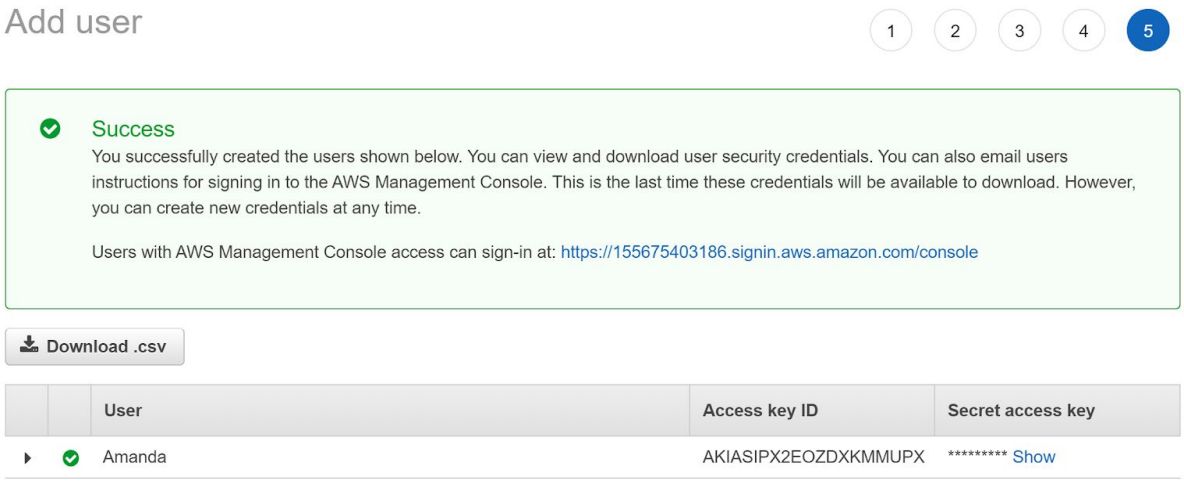


Figure 48

In a separate tab open Github and navigate back to the settings.py file we used earlier. Click the pencil to edit again.

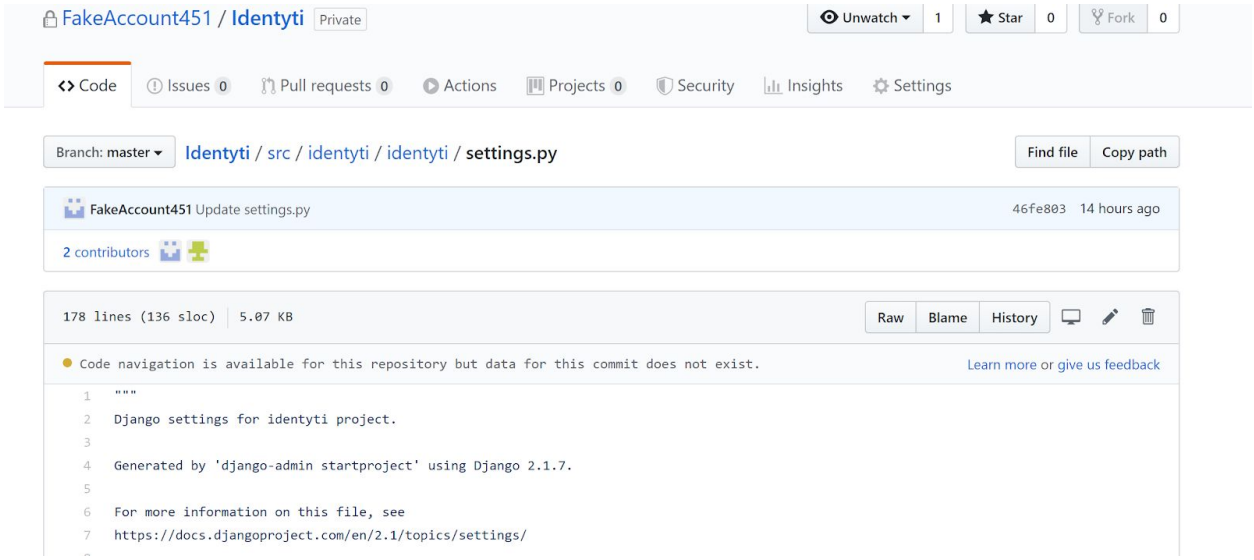


Figure 49

Find the lines that have AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, and AWS_STORAGE_BUCKET_NAME.

```
137
138
139  # Static files (CSS, JavaScript, Images)
140  # https://docs.djangoproject.com/en/2.1/howto/static-files/
141  STATIC_ROOT = os.path.join(BASE_DIR, 'static')
142  #STATICFILES_DIRS = [
143  #    os.path.join(BASE_DIR, 'static'),
144  #]
145
146  #Configuring s3 bucket to dump static and media file data
147  AWS_ACCESS_KEY_ID = 'AKIAWCX62H57YSBGL2OA'
148  AWS_SECRET_ACCESS_KEY = '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'
149  AWS_STORAGE_BUCKET_NAME = 'identyti-media'
150  AWS_S3_CUSTOM_DOMAIN = '%s.s3.amazonaws.com' % AWS_STORAGE_BUCKET_NAME
151  AWS_DEFAULT_ACL = 'private'
152
153  AWS_S3_OBJECT_PARAMETERS = {
154      'CacheControl': 'max-age=86400',
155  }
```

Figure 50

Replace the AWS_ACCESS_KEY_ID value with the AWS access key ID for your user, replace AWS_SECRET_ACCESS_KEY with the AWS secret access key for your user, and AWS_STORAGE_BUCKET_NAME with the name of the bucket we created earlier.

After you've made these changes click "Commit changes".

You've successfully completed the AWS set up! Yay!

3.6.6 Setting up email with SendGrid

From your app dashboard on Heroku, choose the resources tab and in the search box search for SendGrid. Next click on SendGrid and you will be redirected to the SendGrid Dashboard. You may be prompted to "Verify your account" by Heroku. This will involve you adding a credit card. Heroku should only charge you a test-charge. After this verification return to https://app.sendgrid.com/ .

If prompted, enter an email address to send a confirmation email to. This is so sendgrid can contact you if needed.

From the dashboard select Click "Sender-Whiz" and then Setup guide.

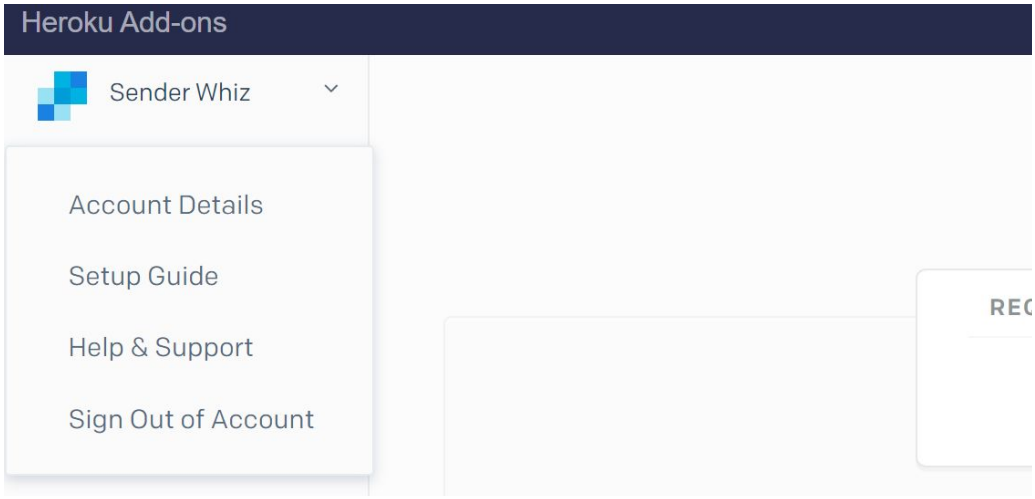


Figure 51

Click “Integrate using our Web API or SMTP Relay” then “SMTP Relay.” Name your api key whatever you would like. Next, copy api keys to settings.py .

This is what the API keys will look like on SendGrid:

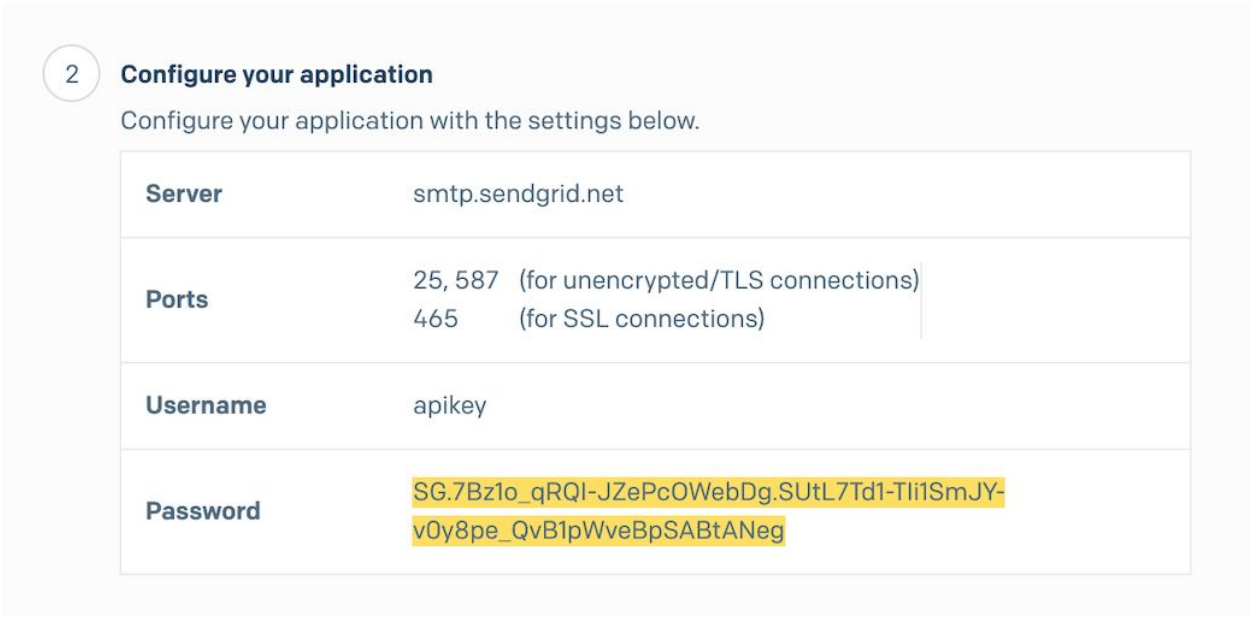


Figure 52

This is what they should look like in the settings.py file:

```
EMAIL_HOST = 'smtp.sendgrid.net'
EMAIL_PORT = 587
EMAIL_HOST_USER = 'apikey'
EMAIL_HOST_PASSWORD = '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'
EMAIL_USE_TLS = True
```

Figure 53

Your application should now be ready to send emails.

3.6.7 Setting up Tesseract for local testing

In order to test the machine learning model locally you first need to download Google's Tesseract OCR. Navigate to this link, https://github.com/tesseract-ocr/tesseract/wiki, and download the necessary files for your operating system. For example:

If you are a windows user, you would download this executable file:

https://digi.bib.uni-mannheim.de/tesseract/tesseract-ocr-w64-setup-v5.0.0-alpha.20200328.exe

and go through the setup process. The tesseract executable file should be located in `C:\Program Files\Tesseract-OCR\tesseract.exe` or somewhere similar.

If you are a mac user, you would need to run the command `brew install tesseract` ,then your tesseract file should be located in `/usr/local/Cellar/tesseract/3.05.02/share/tessdata/` or a similar location.

After downloading tesseract, find the path to the executable file as shown above and copy that location. Next, navigate to the settings.py file in Identyti. You will see this line near the top:

```
# Build paths inside the project like this: os.path.join(BASE_DIR, ...)

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

if 'TRAVIS' not in os.environ:

    pytesseract.pytesseract.tesseract_cmd = r'/app/.apt/usr/bin/tesseract'
```

Change it like so:

```
# Build paths inside the project like this: os.path.join(BASE_DIR, ...)

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

if 'TRAVIS' not in os.environ:

    pytesseract.pytesseract.tesseract_cmd = r'{your path here}'
```

Replace {insert your path here} with the path you found in the previous step (remember to remove the brackets). Similarly, in the middle of settings.py you will find this:

```
else:

    DATABASES = {

        'default': {

            'ENGINE': 'django.db.backends.sqlite3',

            'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),

        }

    }

    if 'TRAVIS' not in os.environ:

        pytesseract.pytesseract.tesseract_cmd =
r'/app/.apt/usr/bin/tesseract'
```

Here, do the same thing as before and change the value of pytesseract.pytesseract.tesseract_cmd to r'{your path here}', where {your path here} is the path you found two steps ago.

After this you should be able to test the machine learning model locally.

IMPORTANT NOTE:

When deploying to Heroku change these paths back to /app/.apt/usr/bin/tesseract so it works on the publicly accessible site.

4. Results

The early stages of starting a business are not that different from a science experiment. A business idea is like a hypothesis; the person starting the business is making an educated guess that people will use his product of service. In order to seek funding for this business, they need to prove to some degree that this hypothesis is correct. This is where a system like the one built for this capstone comes into play. It is too expensive to build the entire system before knowing with a certain level of confidence whether or not it will succeed, but the Identyti web application built for this capstone was relatively inexpensive to build. This application provides an interface in which "users can simply scan & upload their documents - and then access them easily for future use without having to worry about finding them or losing them" (Henderson 2020). This system can prove the hypothesis: people will be willing to store their personal documents online if there is a clear benefit of convenience for them.

For Identyti founder, James Henderson, the benefit is clear: "The system solves the primary problem of creating a way to systematically and efficiently organize all important user documents, while also enabling enterprise clients to issue digital documents directly to users" Henderson said in a phone interview. Identyti has the potential to change the way people use and acquire documents, saving them trips to the DMV and the stress of searching through stacks of paper, "by automatically labeling and sorting user docs, the system condensed an average task of 4 hours to 15 minutes" and "alleviates user anxiety that they may not be able to find their important documents when they need them" (Henderson 2020).

5. Conclusions

Identyti will change the way people use and obtain personal documents if the company is able to get off of the ground. The need has already been identified; the current way that people store personal documents, paper copies, is inconvenient, especially for those who move around a lot. Mailing these documents or gathering them up in order to make a trip to the DMV is inconvenient and nobody enjoys doing it. Identyti just needs to demonstrate that people are willing to use an internet hosting service to store their documents. Once this has been proven, Identyti can expand in order to solve this problem for everyone in the US.

With Identyti used around the US, there would be far-reaching impacts through many industries. First, government regulation through documents like passports, driver's licenses, and social security numbers would be more streamlined. Wherever these documents are needed could be made digital, accelerating the slow processes that are currently in place. For example, the long, slow lines at the DMV could be made much faster using Identyti to store, share, and edit government documents. Identyti may even eliminate the need to go to these establishments in-person, as major enterprises can share, request, and receive important documents directly from the site. Businesses can also improve their hiring and payment processes through Identyti, automatically checking personal documents and sending pay stubs through Identyti. There are an endless number of use cases. With services like Identyti, we take one step closer from a slow and inefficient system to a more secure and streamlined world.

6. Future Work

The next step is for Henderson to start looking for customers in order to prove the business idea. Once he has acquired enough users, he will start looking for funding. This funding will make it possible to hire developers who will likely start building a better version of our app from scratch. That is not to say that our app will not play a valuable role in the growth of this business; this application will give the idea a chance to be tested. Data can also be collected from users of our application in order to help make decisions associated with the building of the new application. The application that was built over the past year provides an important tool to gauge user interest and prototype a final product. In the future, this application can be further developed to accommodate larger scale use and test more features. In order for this project to succeed in the future, a consistent user base must be found. Once there is a solid foundation of users, feedback from these users can be gathered to improve upon the current idea. With these improvements, more users would join and the cycle would repeat itself. Through this process, the application will gain in popularity and grow more successful.

7. References

Biddle, P. (2017, July 26). *Productivity, Lost Time, and the Power of AI to Make Search Easier.* Retrieved from Medium: https://medium.com/@diamond_io/productivity-lost-time-and-the-power-of-ai-to-make-search-easier-a59d4cd85a26

Henderson, J. (2020, March 24). Phone interview.

Klosowski, T. (2013, January 8). *How to Get Through the DMV with your Sanity Intact*. Retrieced from Lifehacker: https://lifehacker.com/how-to-get-through-the-dmv-with-your-sanity-intact-5974078

Larson, G. (2019). *Verifying Identity In Today's Digital Economy: A Look At Regulated Retail.* Retrieved from Forbes: https://www.forbes.com/sites/forbestechcouncil/2019/09/06/verifying-identity-in-todays-digital-economy-a-look-at-regulated-retail/#24546f551e62

Matsakis, L. (2019). *The WIRED Guide to Your Personal Data (and Who Is Using It)*. Retrieved from WIRED: https://www.wired.com/story/wired-guide-personal-data-collection/

Poza, D. (2018, July 19). *How Auth0 Makes Your Apps More Secure.* Retrieved November 20, 2019, from Auth0: https://auth0.com/blog/how-auth0-makes-your-apps-more-secure/