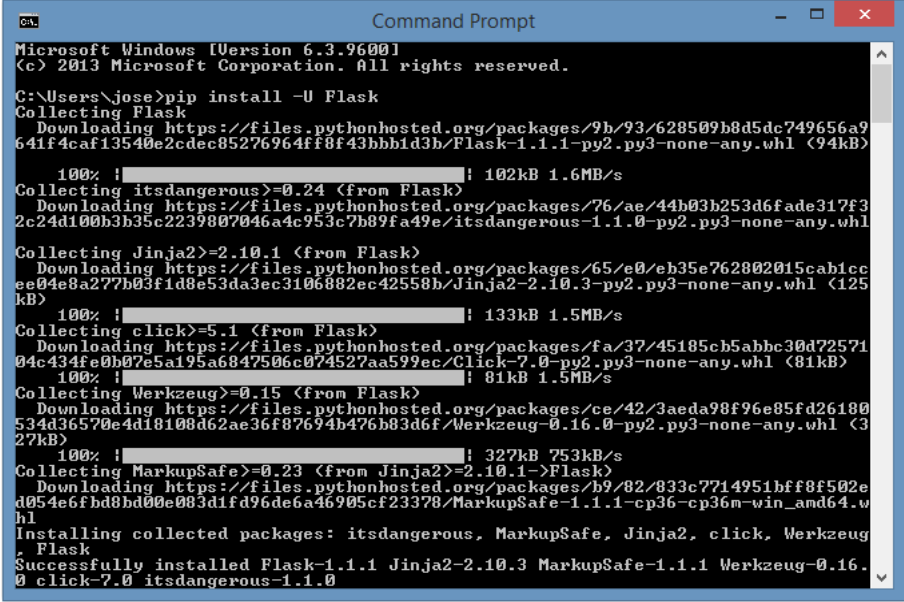


The following document will show how to deploy a simple database into production using open source software and packages. It will first describe the procedure on how to set up and locally run a Flask environment. Then how to use git for adding files to a repository. Finally, how to do a web deployment with Heroku.

Step 1. Flask Installation

This step requires that the user already has Python installed in their computer, it assumes that the user also has installed Pip. To install flask we open the command line and type "pip install -U Flask", the installation begins immediately and we receive a prompt that flask was successfully installed.



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\jose>pip install -U Flask
Collecting Flask
  Downloading https://files.pythonhosted.org/packages/9b/93/628509b8d5dc749656a9641f4caf13540e2cdec85276964ff8f43bbb1d3b/Flask-1.1.1-py2.py3-none-any.whl (94kB)
    100% |#####| 102kB 1.6MB/s
Collecting itsdangerous>=0.24 (from Flask)
  Downloading https://files.pythonhosted.org/packages/76/ae/44b03b253d6fade317f32c24d100b3b35c2239807046a4c953c7b89fa49e/itsdangerous-1.1.0-py2.py3-none-any.whl
Collecting Jinja2>=2.10.1 (from Flask)
  Downloading https://files.pythonhosted.org/packages/65/e0/eb35e762802015cab1cc04e8a277b03f1d8e53da3ec3106882ec42558b/Jinja2-2.10.3-py2.py3-none-any.whl (125kB)
    100% |#####| 133kB 1.5MB/s
Collecting click>=5.1 (from Flask)
  Downloading https://files.pythonhosted.org/packages/fa/37/45185cb5abbc30d7257104c434fe0b07e5a195a6847506c074527aa599ec/Click-7.0-py2.py3-none-any.whl (81kB)
    100% |#####| 81kB 1.5MB/s
Collecting Werkzeug>=0.15 (from Flask)
  Downloading https://files.pythonhosted.org/packages/ce/42/3aeda98f96e85fd26180534d36570e4d18100d62ae36f87694b476b83d6f/Werkzeug-0.16.0-py2.py3-none-any.whl (327kB)
    100% |#####| 327kB 253kB/s
Collecting MarkupSafe>=0.23 (from Jinja2>=2.10.1->Flask)
  Downloading https://files.pythonhosted.org/packages/b9/82/833c7714951bffa8f502ed054e6fbd8bd00e083d1fd96de6a46905cf23378/MarkupSafe-1.1.1-cp36-cp36m-win_amd64.whl
Installing collected packages: itsdangerous, MarkupSafe, Jinja2, click, Werkzeug, Flask
Successfully installed Flask-1.1.1 Jinja2-2.10.3 MarkupSafe-1.1.1 Werkzeug-0.16.0 click-7.0 itsdangerous-1.1.0
```

Figure 1: Installing Flask from command line

Step 2. Running Flask

First we create a new folder called "server" in the desktop. Then, we use the text editor "Atom" to create a new file called "app.py" that will be located within the new folder. The file contains the code provided within the assignment in question 4.

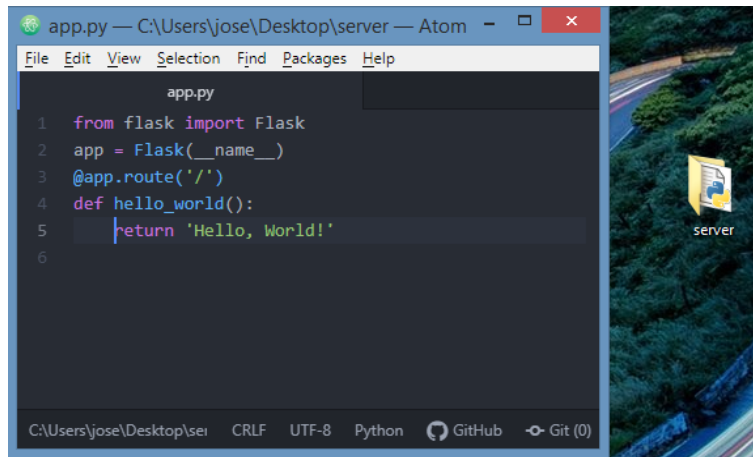


Figure 2: Creating an app.py file in server folder

We then go back to the command line and execute the file by following these steps: First we call down to the server file. Then we set an environmental variable for flask using "set FLASK_APP=app". Finally, we use "flask run". This sets up the environment server and we can check the IP address at the end to verify its working. As shown in the following figure, we run the code and open a web browser to the page <http://127.0.0.1:5000> and confirm its working accordingly.

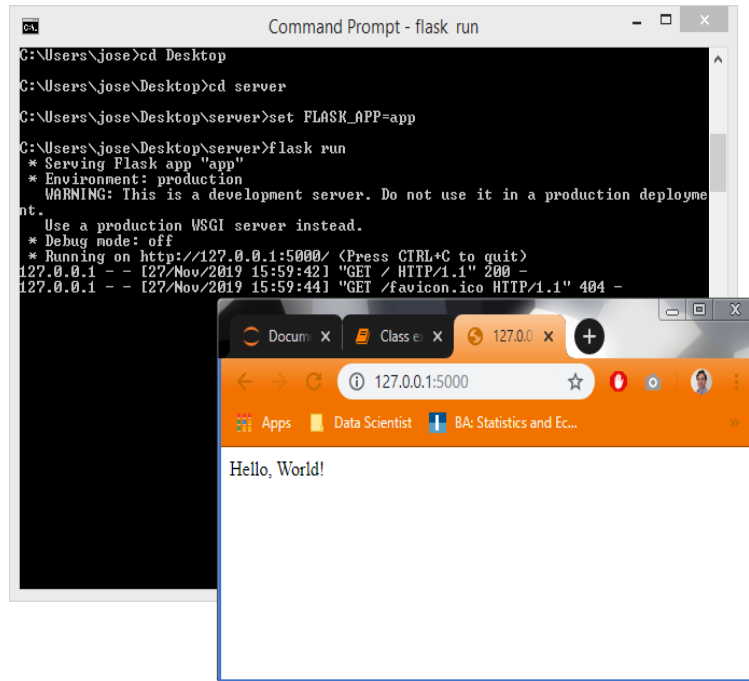


Figure 3: Running the app.py file with Flask

Step 3. Running Flask

We go back and change the text of the file app.py from "Hello, world!", to "Example code for question 5". We refresh the web page and notice that that the new text values are not shown.

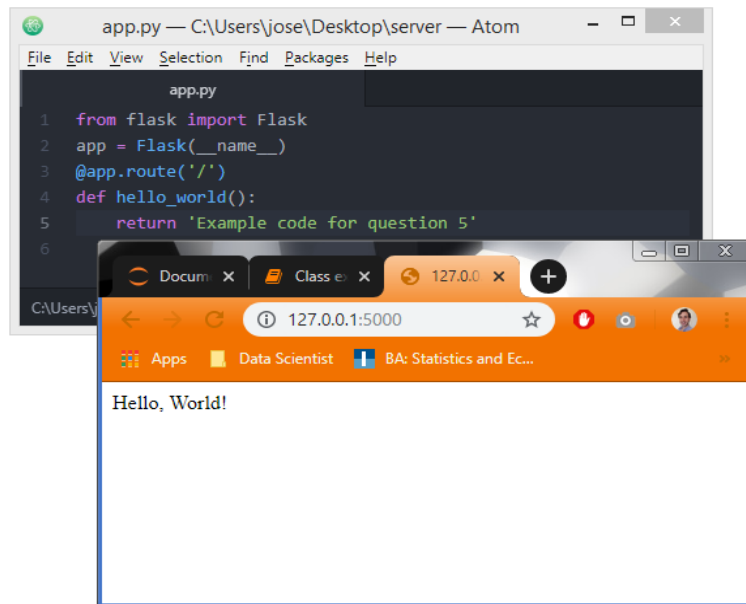
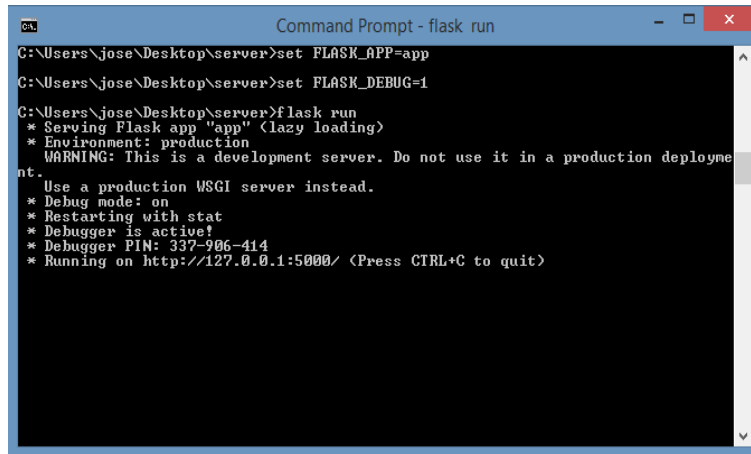


Figure 4: Changing code from app.py file and refreshing webpage

To have the changes be reflected on the web page we need to configure our Flask environment to automatically reload the code when it is changed. Again, we set an environmental variable for flask using "set FLASK_APP=app". We will add a new command for this occasion "set FLASK_DEBUG=1", and run the code again using "flask run".



```
C:\Users\jose\Desktop\server>set FLASK_APP=app
C:\Users\jose\Desktop\server>set FLASK_DEBUG=1
C:\Users\jose\Desktop\server>flask run
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 337-906-414
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figure 5: Changing Flask to automatically reload the code

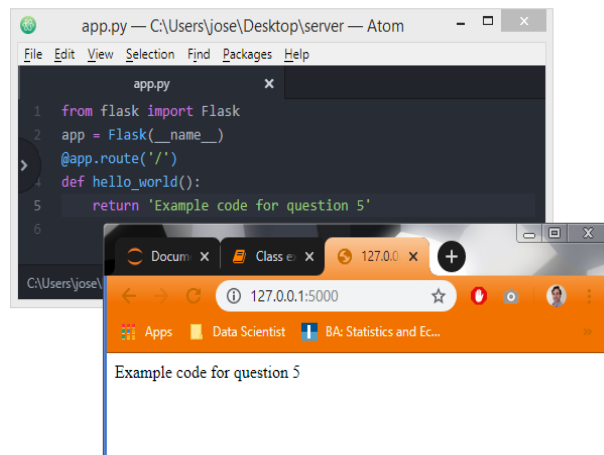
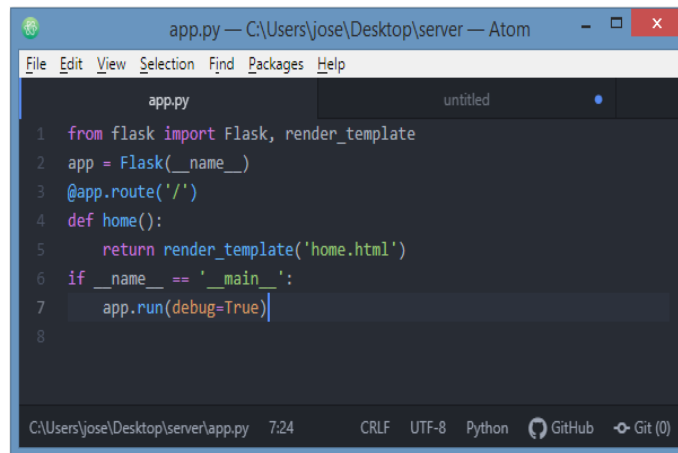


Figure 6: Verifying that code refreshes automatically on webpage

Step 4. Adding templates to Flask app

Next, we will add a template to your Flask app. We follow the instructions from the webpage "<https://pythonhow.com/html-templates-in-flask/>". In this tutorial we will use the flask "render_template" method. We commence by modifying our code as shown in the following figure.



```
app.py — C:\Users\jose\Desktop\server — Atom
File Edit View Selection Find Packages Help

app.py
1 from flask import Flask, render_template
2 app = Flask(__name__)
3 @app.route('/')
4 def home():
5     return render_template('home.html')
6 if __name__ == '__main__':
7     app.run(debug=True)
8

C:\Users\jose\Desktop\server\app.py 7:24 CRLF UTF-8 Python GitHub Git (0)
```

Figure 7: Using FLASK render_template method

Then we follow by creating a new a new file called "home.html".

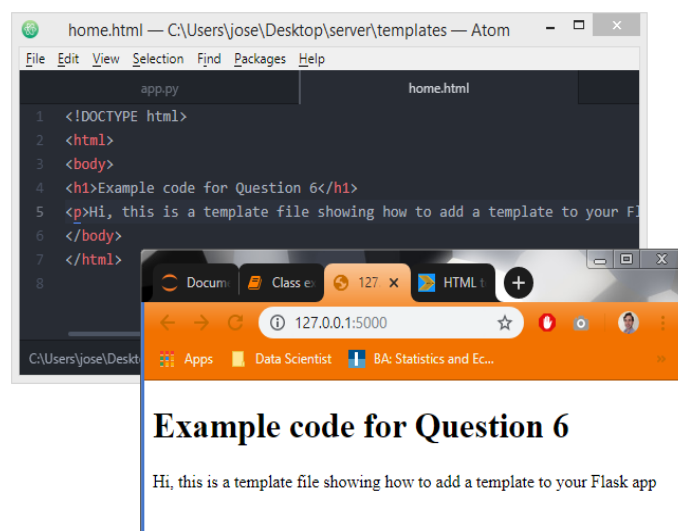


Figure 8: Showing a web page using Flask templates

Step 5. Create a git repository

First we install git. We are using the Windows version of git, which was downloaded from the following link: "<https://git-scm.com/download/win>"

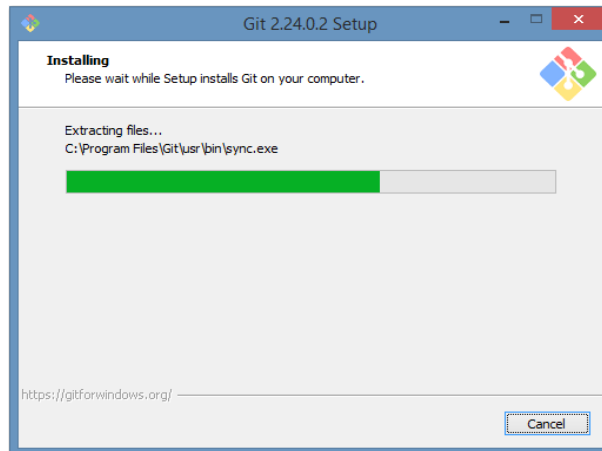


Figure 9: Installation of Git

After the installation we call down to the server folder and initialize a git session with the command "git init".

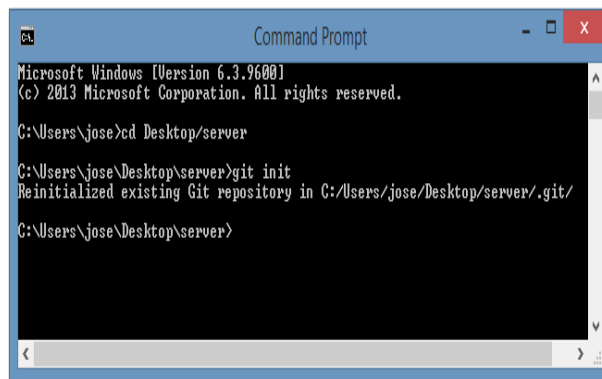
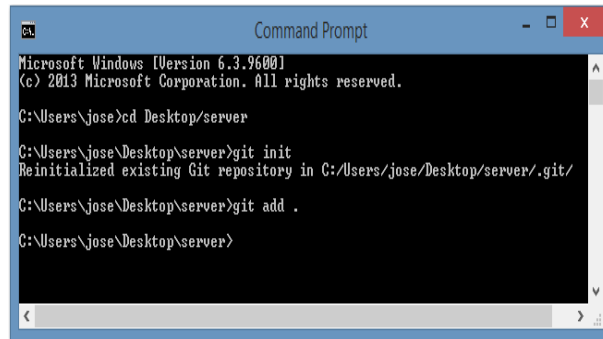


Figure 10: Using the command Git init to initialize a session

Step 6. Adding files to git repository

The next step is to add all the files from the folder to the git session. We do this by calling the command "git add .", the "." is to specify all the files in the folder. Otherwise we have to individually add each file.



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\jose>cd Desktop/server

C:\Users\jose\Desktop\server>git init
Reinitialized existing Git repository in C:/Users/jose/Desktop/server/.git/

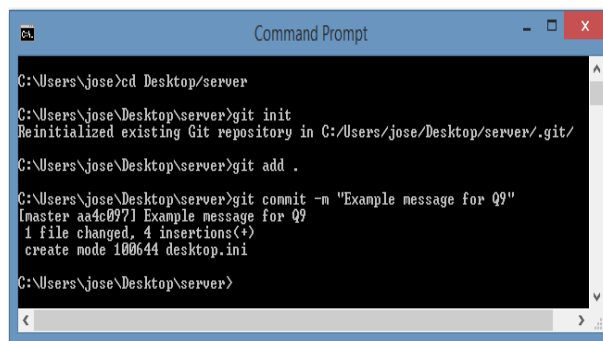
C:\Users\jose\Desktop\server>git add .

C:\Users\jose\Desktop\server>
```

Figure 11: Using the "git add" command

Step 7. Committing files to git repository

We finalize with the statement "git commit", additionally we can use the command "-m" to add a message in quotes, this is useful to have documentation about the commit history.



```
C:\Users\jose>cd Desktop/server

C:\Users\jose\Desktop\server>git init
Reinitialized existing Git repository in C:/Users/jose/Desktop/server/.git/

C:\Users\jose\Desktop\server>git add .

C:\Users\jose\Desktop\server>git commit -m "Example message for Q9"
[master aa4c097] Example message for Q9
1 file changed, 4 insertions(+)
create mode 100644 desktop.ini

C:\Users\jose\Desktop\server>
```

Figure 12: Using the "git commit" command

Step 8. Signing up to Heroku

We sign up for a new account at the Heroku webpage: "<https://signup.heroku.com/>" and install the software in the computer.

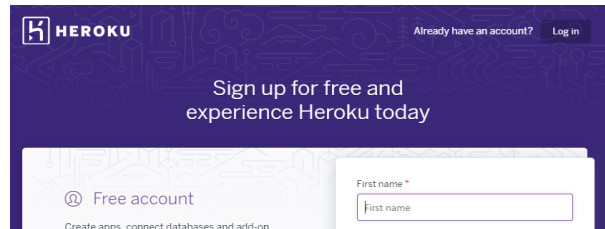


Figure 13: Signing up for a Heroku account

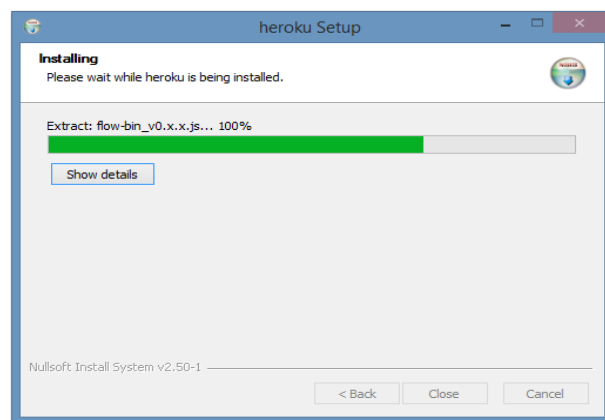


Figure 14: Installation of Heroku toolbelt

Step 9. Deploying a Heroku app

First we verify the correct installation of Heroku with the command "heroku --version". Next we call down to the server folder and use the command "heroku login". This take us to the heroku website to sign in. After a succesful sign in, we can receive confirmation in the command prompt that we are logged in.

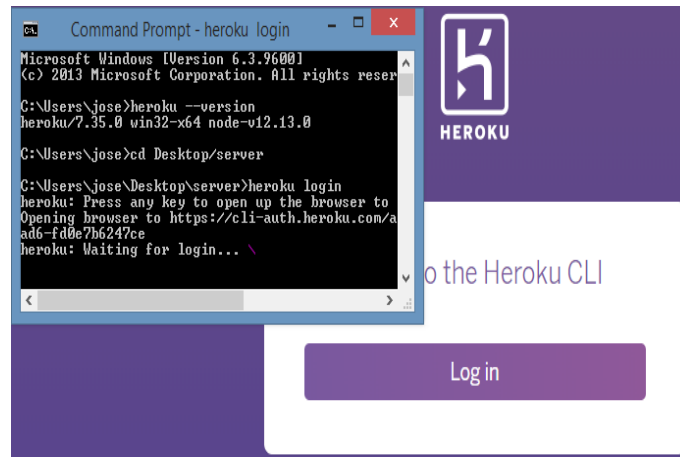


Figure 15: Log in to Heroku account through webpage

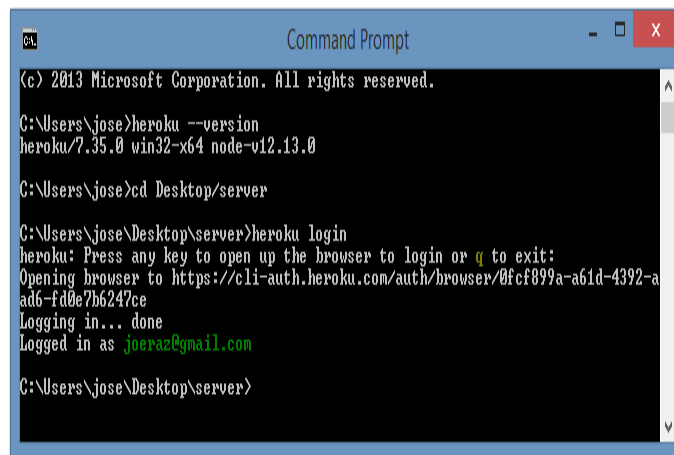
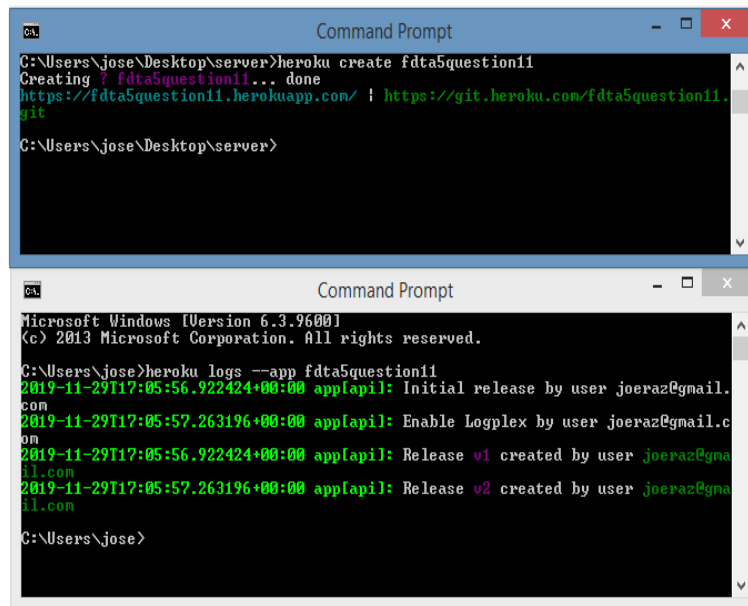


Figure 16: Log in complete, command prompt confirmation

To create a heroku app, we use the command "heroku create fdta5question11", and open a second windows where we can check the logs of the app using the command "heroku logs --app fdta5question11". Additionally we can check the heroku account on the website and confirm that a heroku app has been created.



```
C:\Users\jose\Desktop\server>heroku create fdta5question11
Creating ? fdta5question11... done
https://fdta5question11.herokuapp.com/ | https://git.heroku.com/fdta5question11.git
C:\Users\jose\Desktop\server>

C:\Users\jose>heroku logs --app fdta5question11
2019-11-29T17:05:56.922424+00:00 app: Initial release by user joeraz@gmail.com
2019-11-29T17:05:57.263196+00:00 app: Enable Logplex by user joeraz@gmail.com
2019-11-29T17:05:56.922424+00:00 app: Release v1 created by user joeraz@gmail.com
2019-11-29T17:05:57.263196+00:00 app: Release v2 created by user joeraz@gmail.com
C:\Users\jose>
```

Figure 17: Creating Heroku app

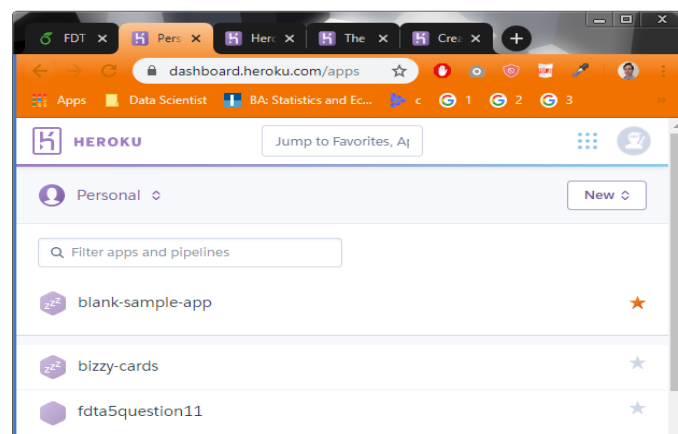


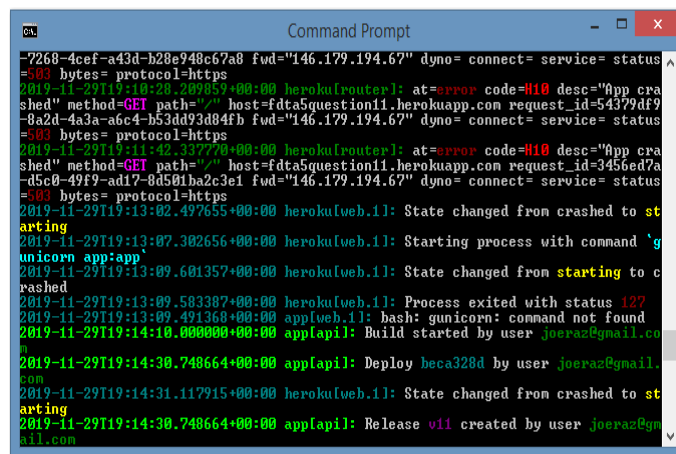
Figure 18: Verifying heroku app has been created

After reading the documentation for deployment to the heroku web service, we had to create extra files and git commit the files to perform a successful deployment.

To continue the deployment of the heroku app several steps had to followed in the following order.

1. Installation of the following packages using the command prompt: FLASK.SQLAlchemy with command "pip install flask sqlalchemy; SQLAlchemy with "pip install sqlalchemy"; Psycopg2 with "pip install psycopg2"; and Gunicorn with "pip install gunicorn"
2. A requirements.txt file was created to specify the software that the app needs: the list included is "flask, flask-sqlalchemy, psycopg2, and gunicorn".
3. A procfile was created with the contents "web: gunicorn app:app".
4. A runtime.txt file specifying the version of python begin used, in this case "python-3.6.0"
5. Repetition of git commands, in the following order: Initializing a session, adding all files to session, committing, and finally using a new commmand git "push", explanation follows:
6. Deploying the code with the command "git push heroku master", this pushes the code up to the heroku web service, heroku takes all the files and runs the code on the web to create our website.

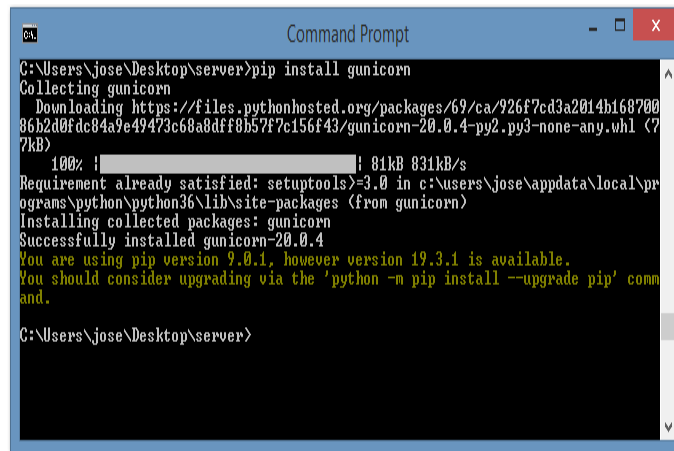
Before deployment, a few bugs were incurred. After several modification to the procedures the solution was the installation of package "gunicorn", which was missing from the specified components in the deployment.



```

-7268-4cef-a43d-b28e948c67a8 fwd="146.179.194.67" dyno= connect= service= status
=503 bytes= protocol=https
2019-11-29T19:10:28.209859+00:00 heroku[router]: at=error code=H10 desc="app cra
shed" method=GET path="/" host=fdta5question11.herokuapp.com request_id=54379df9
-8a2d-4a3a-a6c4-b53dd93d84fb fwd="146.179.194.67" dyno= connect= service= status
=503 bytes= protocol=https
2019-11-29T19:11:42.337770+00:00 heroku[router]: at=error code=H10 desc="app cra
shed" method=GET path="/" host=fdta5question11.herokuapp.com request_id=3456ed7a
-d5c0-49f9-ad17-8d501ba2c3e1 fwd="146.179.194.67" dyno= connect= service= status
=503 bytes= protocol=https
2019-11-29T19:13:02.497655+00:00 heroku[web.1]: State changed from crashed to st
arting
2019-11-29T19:13:07.302656+00:00 heroku[web.1]: Starting process with command 'g
unicorn app:app'
2019-11-29T19:13:09.601357+00:00 heroku[web.1]: State changed from starting to c
rashed
2019-11-29T19:13:09.583307+00:00 heroku[web.1]: Process exited with status 127
2019-11-29T19:13:09.491368+00:00 app[web.1]: bash: gunicorn: command not found
2019-11-29T19:14:10.000000+00:00 app[api]: Build started by user joeraz@gmail.co
m
2019-11-29T19:14:30.748664+00:00 app[api]: Deploy beca328d by user joeraz@gmail
.com
2019-11-29T19:14:31.117915+00:00 heroku[web.1]: State changed from crashed to st
arting
2019-11-29T19:14:30.748664+00:00 app[api]: Release v11 created by user joeraz@gm
ail.com
```

Figure 19: Bug Issues during installation



```
C:\Users\jose\Desktop\server>pip install gunicorn
Collecting gunicorn
  Downloading https://files.pythonhosted.org/packages/69/ca/926f7cd3a2014b168700
86b2d0fd84a9e49473c68a8dff8b57f7c156f43/gunicorn-20.0.4-py2.py3-none-any.whl (7
7kB)
    100% |#####| 81kB 831kB/s
Requirement already satisfied: setuptools>=3.0 in c:\users\jose\AppData\Local\pr
ograms\python\python36\lib\site-packages (from gunicorn)
Installing collected packages: gunicorn
Successfully installed gunicorn-20.0.4
You are using pip version 9.0.1, however version 19.3.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' comm
and.

C:\Users\jose\Desktop\server>
```

Figure 20: Installation of necessary packages to complete deployment

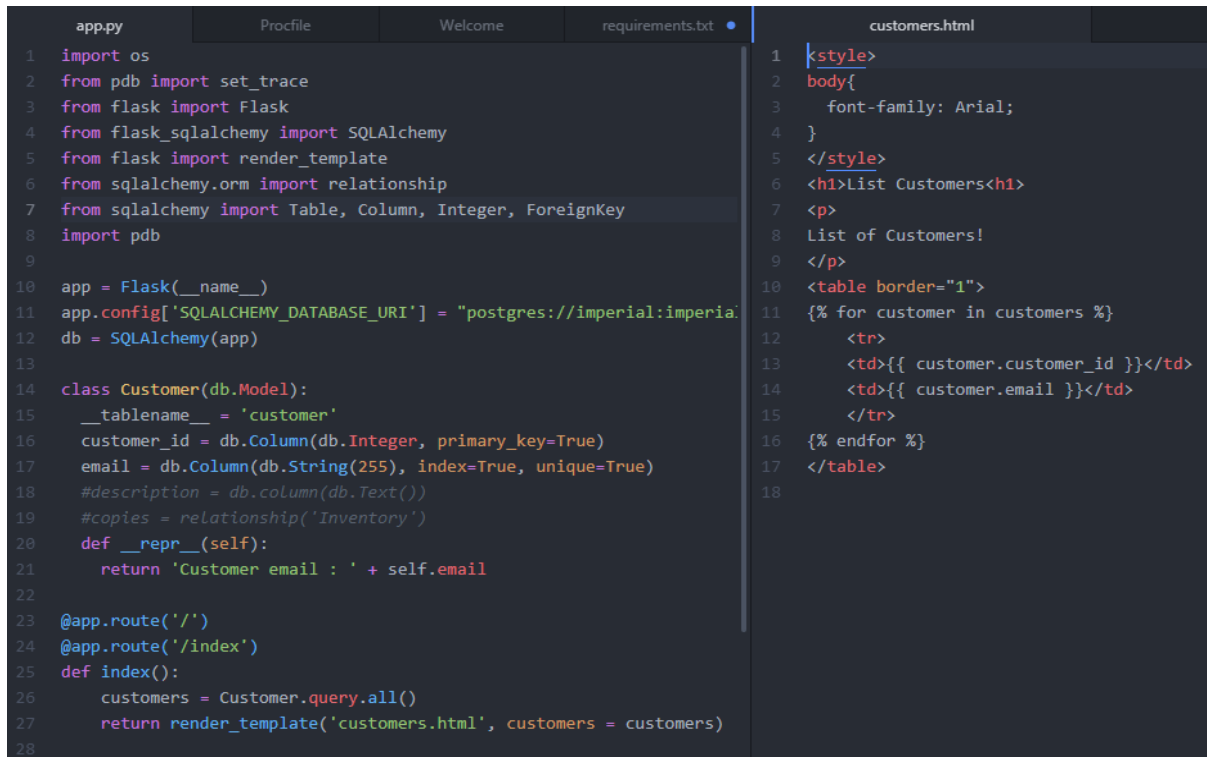
If all the steps are followed correctly then the app is correctly deployed to the Heroku service. The following figure shows a visit to the app "fdta5question11" on the Heroku web address to demonstrate the proper deployment.



Figure 21: Showing a completed heroku deployment on the web

Step 10. Creatig page listing of all the customers

Drawing on the Flask sample code made available on the Imperial College Hub, we add a database model from the "dvdrental" database. First we will run the app locally, the following figure shows a screen shot of the code on the text editor "Atom", the purpose of this page listing is to extract all the customer names and emails to show on the web site.



```
app.py
1 import os
2 from pdb import set_trace
3 from flask import Flask
4 from flask_sqlalchemy import SQLAlchemy
5 from flask import render_template
6 from sqlalchemy.orm import relationship
7 from sqlalchemy import Table, Column, Integer, ForeignKey
8 import pdb
9
10 app = Flask(__name__)
11 app.config['SQLALCHEMY_DATABASE_URI'] = "postgres://imperial:imperial@localhost:5432/dvdrental"
12 db = SQLAlchemy(app)
13
14 class Customer(db.Model):
15     __tablename__ = 'customer'
16     customer_id = db.Column(db.Integer, primary_key=True)
17     email = db.Column(db.String(255), index=True, unique=True)
18     #description = db.Column(db.Text())
19     #copies = relationship('Inventory')
20     def __repr__(self):
21         return 'Customer email : ' + self.email
22
23 @app.route('/')
24 @app.route('/index')
25 def index():
26     customers = Customer.query.all()
27     return render_template('customers.html', customers = customers)
28
```

```
customers.html
1 <style>
2 body{
3     font-family: Arial;
4 }
5 </style>
6 <h1>List Customers</h1>
7 <p>
8 List of Customers!
9 </p>
10 <table border="1">
11     {% for customer in customers %}
12     <tr>
13     <td>{{ customer.customer_id }}</td>
14     <td>{{ customer.email }}</td>
15     </tr>
16     {% endfor %}
17 </table>
18
```

Figure 22: Code used for local deployment

In the following figure, we can see that the app is running locally using the same procedure we had with a Flask environment. We are running it on a local server to quickly iterate through any problems and have a preview of how it will look once deployed.

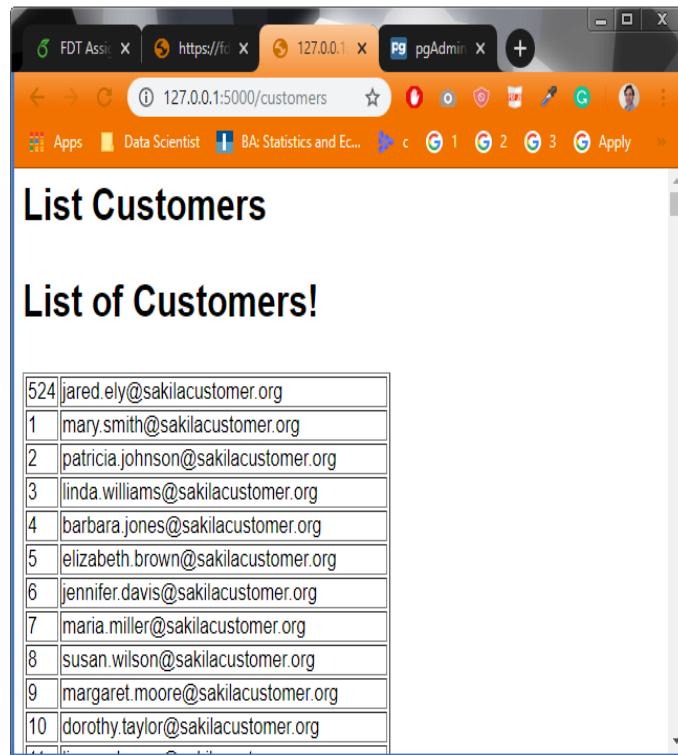


Figure 23: Customer page listing set in FLASK environment

Step 11. Deploying new version to Heroku

For the final step we deploy the new version of the app to Heroku. We follow the same procedure as in step 11. We add all the files to the git commit and push them with the command "git push heroku master". If all the steps are followed correctly then can the result as in the following figure. To check this website is up and running, please visit "https://fdta5question11.herokuapp.com/".

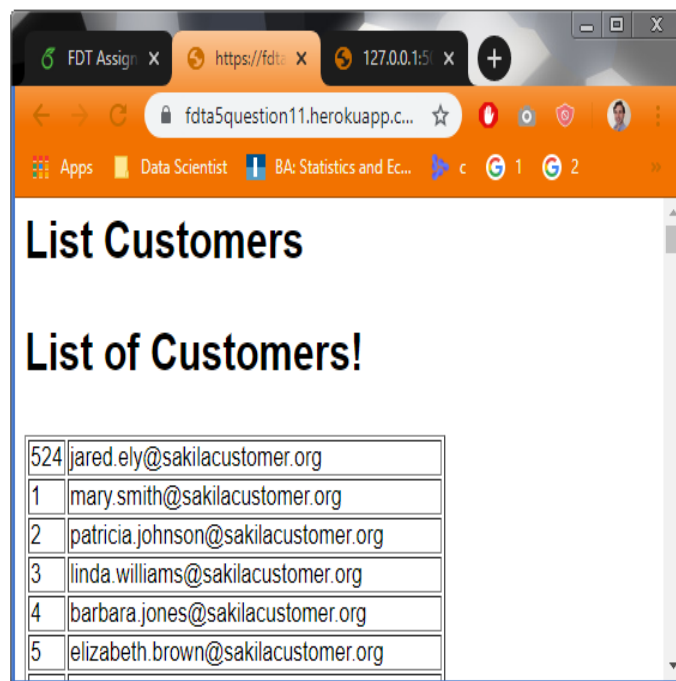


Figure 24: Deployed final version of the app unto the Heroku service

In conclusion, the document showed some key packages for quickly and efficiently deploying a web service. We used a Flask environment to set up a local server. Then we reviewed basic git commands, and finally we showed the procedure of how to set up a Heroku web deployment that is connected to a database.