CA4106 Cloud Computing
Group Project

By
Alex Preston - 16439734
Eoin Dunne - 16408544
Eoin Girvin -

Planning/Research:

- After careful consideration and planning we finally decided to go ahead and start working on making a blogs app through Django.
- We chose to use Django because we have previous experience using it and knew it integrated well with SQL.
- We watched a number of YouTube tutorials to investigate what kind of features we could implement through the use of Django.
- We decided on making the blogs about the Coronavirus and how it will have an impact on students and teachers/professors all over the world.
- We chose to do this because it's a topical subject and it's affecting everyone around the world.
- Users will be able write, edit and delete blogs written on the web app.
- We also wanted to show a tracking "map" of a global view of the coronavirus and how many cases are currently in each country. To do this we could use a widget imported from another website.
- We also decided to use a Google Doc so that we could all participate simultaneously on the document rather than constantly pushing and pulling from Git to work on it. We chose to do this because we have run into trouble with Git merging in the past and we felt this was more suitable.
- To help with planning and collaboration we also kept track of our progress with the use of a Trello board which we all had access to, and when we completed a task or found a bug we could log it for everyone to see.

Trello Board For Timekeeping

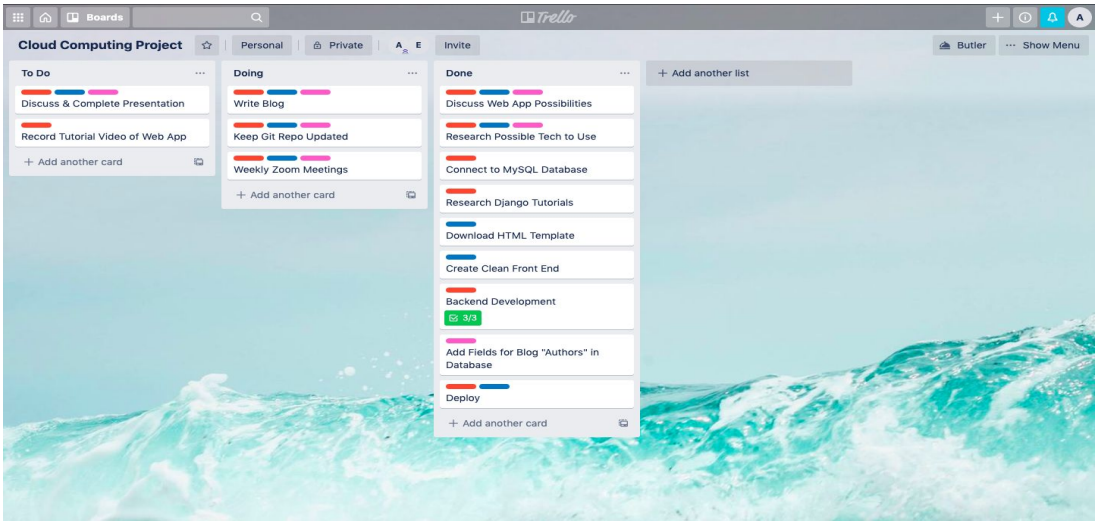
Link to Trello Board:

https://trello.com/invite/b/8eNVqBIE/2f93652f64dbc3152a4f321f9b3a9582/cloud-computing-project

We added a Trello board to keep track of what each person was doing and to see when each task was completed. We added different coloured labels for each individual in the group. From the labels on the board you can see that we worked together on most of the project bar one or two tasks which we took upon us individually.

Label Colours:

Alex Preston = Blue Label
Eoin Dunne = Orange Label
Eoin Girvin = Pink Label



Technologies Used:

- Google Cloud Platform
- HTML
- CSS
- JavaScript
- Python
- MySQL
- Google SDK
- Django Framework

Front End Development:

- Alex was responsible for most front end productivity. We have all built numerous web apps/websites but Alex felt most comfortable with the front end side of things.
- We met on Zoom to discuss what theme we wanted for our web app and how we should go about styling it.
- We decided on using a template from w3schools which used Bootstrap CSS for styling. We decided to do this because Bootstrap allowed us to create a clean and concise website without having to spend hours working on styling, and because this project is more orientated around connecting to a SQL database and uploading the project to Google cloud services.
- We decided to use the Django framework which meant that we had to add in certain “tags” such as “{% load static %} which allowed us to add images to the web pages. This caused some issues at the start but after searching online for solutions we eventually fixed the issue with the help of Stack Overflow.

Backend Development

- Eoin Dunne was responsible for the back end development.
- Backend development involved the integrating of the following into the development of our web application.

    - Setting up Django Framework.
    - Creating MySQL Instances within Google Cloud Platform.
    - Building the Blogs application within our application.
    - Initializing our application to the SQL Instances.
    - Configuring Database Settings.
    - Deploying our application to Google Cloud Platform.

Setting up Django Framework - 21st March

Setting up the Django Framework was a vital first step into the development of our web app. We decided to use the Django Framework as it is a completely open-source web framework that is very manageable, clean, and very secure during web development. Django also comes equipped with already developed components so that we could focus more on developing our application rather than spending time developing python files.
Django also enabled us to use core languages within our application such as HTML, CSS, JavaScript, and Python.

1. The first step in setting up our Django Framework was to ensure that the latest version of Python was installed on the machine. We downloaded this from ***https://www.python.org/downloads/***. In this project we used Python Version 3.7.

2. Once we had Python installed the next step was to install the Django Framework. Now that we had Python installed we could use Python functions such as ***pip***. Within the command prompt the function ***"python -m pip install Django"*** was used to install the Django Framework. Now we had the basis to start our development.

Creating MySQL Instance within Google Cloud Platform - 24th March

We used a MySQL Database so that we could store, extract, and edit our data. We researched numerous serverless platforms that could host our MySQL Database. With great thought, we decided to create our SQL Instances on Google Cloud Platform. Google Cloud Platform offered us free hosting, and was very easy to use.

1. Before we created a MySQL Instance the first step was to enable CloudSQL Admin API, as that would allow us to create our database. We did this by first installing and initializing Cloud SDK. We downloaded this from

    https://cloud.google.com/sdk/docs.

2. Next we were to create the Cloud SQL Instance and Database so we can store and extract data. This is a simple process that can be done within the SQL Tab in the Google Cloud Console. We named our Instance **"computing-project"**, and our database **"blogs"**. It was also important to create a User within the Instance. We named ours **"test"**. It was important to keep track of these names as they would be important in future development.

3. Once our Cloud SQL Instance was created, we needed to record this connection name for a future step. This was done by running the following command in the Cloud SDK Console. ***"gcloud sql instances describe computing-project".*** We were returned with the following output in the Console. It was important to take note of the highlighted output as we will use it when connecting Django to the Instance.

```
C:\Users\EoinD\AppData\Local\Google\Cloud SDK>gcloud sql instances describe computing-project
backendType: SECOND_GEN
connectionName: cloud-computing-project-272315:europe-west1:computing-project
databaseVersion: MYSQL_5_7
etag: fa62f382d9f241e314f246128aeb54509112ba4882e0eb3f3a344fee82aff6b9
gceZone: europe-west1-d
```

Building the Back End of The Blogs Application - 27th March

The specifications provided to us for this project were to "Build a web front end to a cloud SQL database". Through researching different SQL applications we found that a Blogs application perfectly utilises SQL Create, Extract, and Delete functions.
The back-end application of the blogs application was built by Eoin and involved the manipulation of python files within the Django Framework that we previously downloaded. The following code was implemented into ***views.py*** to apply these SQL Functions.

```
class PostCreateView(CreateView):
    model = Post
    fields = ['author','title', 'content']
```

```
class PostUpdateView(UpdateView):
    model = Post
    fields = ['title', 'content']
```

```
class PostDeleteView(DeleteView):
    model = Post
    success_url = '/'
```

These functions were then implemented within ***urls.py*** so that the functions were pointing to a certain web page and were easily accessible.

```
path('', PostListView.as_view(), name='blog-home'),
path('post/<int:pk>/', PostDetailView.as_view(), name='post-detail'),
path('post/new/', PostCreateView.as_view(), name='post-create'),
path('post/<int:pk>/update/', PostUpdateView.as_view(), name='post-update'),
path('about', views.about, name='blog-about'),
path('post/<int:pk>/delete/', PostDeleteView.as_view(), name='post-delete')
```

Initialising the Application to the SQL Instance - 1st April

This step establishes a connection from our local computer where our django application is running to our Cloud SQL Instance that we previously created. This is used for testing purposes. In order to run this test we were required to install a Cloud SQL Proxy. We installed this from ***https://dl.google.com/cloudsql/cloud_sql_proxy_x64.exe***

Once this was installed we could run the following prompt from within our Cloud SDK Console. We are using our Connection name that we recorded in a previous step.

```
cloud_sql_proxy.exe -instances="cloud-computing-project-272315:europe-west1:computing-project"=tcp:3307
```

If successful, the following should be displayed showing to us that the connection to our instance was successful.

```
C:\Users\EoinD\AppData\Local\Google\Cloud SDK>cloud_sql_proxy.exe -instances="cloud-computing-project-272315:europe-west
1:computing-project"=tcp:3307
2020/04/07 15:13:16 Listening on 127.0.0.1:3307 for cloud-computing-project-272315:europe-west1:computing-project
2020/04/07 15:13:16 Ready for new connections
```

Configuring the Database Settings - 3rd April

This step was taken to help set up the connection to the database for the App Engine Deployment and for local testing. This was all done within the ***settings.py*** file in the Django Framework. We added our **"Connection Name"** , **"Database Name"** , **"User"**, and **"Password"** that were created in previous steps to initialise the connection.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'HOST': '/cloudsql/cloud-computing-project-272315:europe-west1:computing-project',
        'USER': 'test',
        'PASSWORD': 'test',
        'NAME': 'blogs',
    }
}
```

Once this was completed we could finally run our application to test if everything was working. However, we first needed to create an isolated python environment using ***"py -m venv env"*** then initialising this using ***"env\scripts\activate".***

Once this was created we could run our server to test if it was functioning correctly. We used this by running the following code.
python manage.py makemigrations
python manage.py migrate
python manage.py runserver
We then accessed our localhost on port 8000 to test if this was working and after some troubleshooting it worked perfectly!

Deploying our Application to Google Cloud Platform - 6th April

Once all of our components were integrated into the Django Framework and we had completed our testing on the localhost it was time to deploy our application!
Once we navigated to our django folder that contained all our scripts within the command prompt the following command was used to deploy the application.

"gcloud app deploy"

By default, tests are run within the command prompt to ensure that there are no issues in our application. After some testing and troubleshooting, our application was finally deployed! Once deployed the application is accessible through a URL. Ours was made available on.
https://cloud-computing-project-272315.appspot.com/

Testing Phase - 21st March Onward

- Testing Involved ensuring that after each backup that every commit was working correctly.
- Testing also involved troubleshooting and solving some issues that may be preventing the application from functioning correctly.
- We used testing whenever we ran into any issues such as the problems encountered, discussed in the next section.
- When the SQL Instance was initiated within the SDK console, the SQL database logs were examined to ensure that all components were functioning as expected and that there weren't any error logs visible.
- When the application was running on localhost, the website was tested thoroughly to ensure that all of our implemented features were working flawlessly. This was to create a fluid user experience.
- Lastly, when the application was finally deployed to the google app engine, some intensive, thorough testing was conducted to ensure that all functions were working.
- If there were any issues encountered they would be immediately flagged and we would host a Zoom meeting to ensure that these issues were amended.

Problems Encountered & Solutions

1. The first issue we ran into was during the testing phase when initialising the database to the SQL Instance. The issue that we encountered was when using the command.

```
cloud_sql_proxy.exe-instances="cloud-computing-project-272315:europe-west1:computing-project"=tcp:3306
```

We were returned with the following error message, therefore unable to test that our Instance was actually functioning correctly.

```
C:\Users\EoinD\AppData\Local\Google\Cloud SDK>cloud_sql_proxy.exe -instances="cloud
-computing-project-272315:europe-west1:computing-project"=tcp:3306
2020/04/07 15:49:29 listen tcp 127.0.0.1:3306: bind: An attempt was made to access
a socket in a way forbidden by its access permissions.
```

We attempted changing the port name in the command line however we were still being returned with the same error message. What we didn't realise is that when changing the port number in the command, we are also required to change it in ***settings.py*** also. Once we changed both of the port numbers to **3307** we were able to test our instance without issue and begin listening on port **3307**.

2. In the front end production we ran into an issue with loading the CSS and Javascript in Django. We were missing the {% load static %} tag in our HTML and we were also missing a "*from django.conf.urls.static import static*" in our ***urls.py*** file.

3. We wanted to have an "add" "delete" and "edit" function for the blogs but we had trouble trying to get the "edit" to function. The error we were running into was related to editing the author name, so to fix this issue we removed the ability to change the author of the blog when editing, which we felt was an irrelevant feature.

4. Another error encountered was when attempting to deploy the application to Google Cloud Platform. When the deployment was successful and we attempted to access our web application on the given URL, we were returned with a **502 Bad Gateway** error. This was an error that baffled us for quite a while. We looked into forums where people have had similar issues when deploying their application.

   One user explained to us that they solved their issue by checking the ***main.py*** file in our Django Framework. Sure enough when we checked our ***main.py*** there was a typo in the code. When this typo was fixed and we deployed our fixed application we were forwarded to our web app without any issues!

References:

Youtube Tutorials

- https://www.youtube.com/watch?v=osKPG2ripmw
- https://www.youtube.com/watch?v=-s7e_Fy6NRU

Online widgets for coronavirus stats:

- https://apps.elfsight.com/panel/applications/coronavirus-stats/

Google Cloud Tutorial

- https://cloud.google.com/sql/docs/mysql/connect-app-engine
- https://cloud.google.com/python/django/appengine

HTML & CSS Template (Hyperspace)

- https://html5up.net/