CSE 546 —FREEBIE

Rachana Basabathini (1220231558)
Srikanth Ganji (1220231441)
Nikhil Rayala (1220242829)

1. Introduction

In every college food organizations, a lot of food will get wasted at the end of the day. This food can be given freely to students instead of dumping out. So, we designed a web application that allows college organizations to distribute excess food to students and to search on going events at their university.

The intended target audiences are:

- College students, who are primarily inspired to use this app to locate free food and events in their immediate vicinity.

- College organizations that are inspired to use this app to easily distribute excess food so that it does not go to waste, which has monetary ($ spent on food is not wasted), environmental (food does not end up in landfill), humanitarian (disadvantaged college students receive food), and awareness (organizations will attract more students) benefits.

2. Background

About 20 million students attend 4,000+ degree-granting institutions in the United States. Even though about 80% of students receive financial assistance, researchers discovered that half of them are still 'basic-needs vulnerable,' which involves food insecurity, housing insecurity, and homelessness. Food insecurity was the most common issue among this community, with nearly 40% of respondents agreeing with statements like "I worried if my food would run out before I got money to buy more" and "The food I bought just didn't last and I didn't have money to get more."

On-campus organizations host hundreds of catered events each year, ranging from fast club meetings to multi-day academic conferences and hackathons, belying the reality of many students. While they struggle to obtain reliable access to food, on-campus organizations host hundreds of catered events each year, ranging from quick club meetings to multi-day academic conferences and hackathons. When these activities are finished, the remaining food must be eaten quickly, or it will be discarded. This potential source of food may be directed to those in need, but college organizations which lack the resources to efficiently inform and distribute surplus food, even though students are in an optimal location to obtain it.

Students are not the only ones that suffer from discarded food; the environment suffers as well. This form of potential food waste is the worst in terms of environmental impact: food that has gone through the entire cycle of growing, transportation, storage, and preparation only to end up in the trash.
FREEBIE aspires to close the divide between college organizations and students while also helping the community.

a. **Technologies:**
- **Frontend**: HTML, CSS, JavaScript
- **Backend**: Python
- **Framework**: Django
- **APIs**:
  i. Google Cloud Datastore
  ii. Google Maps JavaScript API
  iii. Gmail API
  iv. Google Geocoding Service

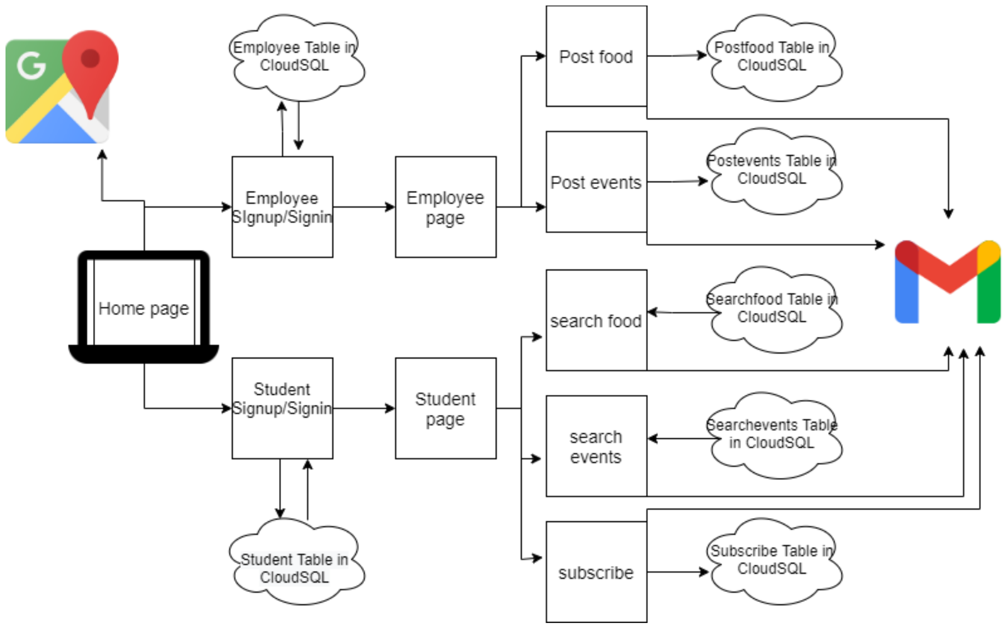
Deployment: Google App Engine

b. **Why is this problem important?**
This problem in crucial as it is about wasting a lot of excess food by college organizations. This food can be given freely to students instead of dumping out. The franchise and the students will both get benefitted with this. As this would be a social service and help students getting free food who cannot afford the meals. And our web application makes it easy to search events at a single platform. This application is not limited to a particular college as any student at any college can register for food/events at their colleges.

c. There are no existing solutions for the food problem.

3. Design and Implementation

a. Architecture


Employee Table in CloudSQL
Post food
Postfood Table in CloudSQL
Employee Signup/Signin
Employee page
Post events
Postevents Table in CloudSQL
Home page
search food
Searchfood Table in CloudSQL
Student Signup/Signin
Student page
search events
Searchevents Table in CloudSQL
Student Table in CloudSQL
subscribe
Subscribe Table in CloudSQL

b. **Cloud Services:**
Google Cloud SQL: This cloud services are used to store, retrieve the data and handling database transactions.
Here we created the following the tables.

employeetable – contains the authentication details of employees.
studenttable - contains the authentication details of students.
postfood – contains the information about food availability posted by the employees of different colleges.
postevent – contains the information about events posted by the employees of different colleges.
subscribe – contains the information about all the students who subscribed.
Google MAPS API: We used this service to get the locations of the colleges n google maps.
Gmail API: We used this service to send mails.

c. Role of Google App Engine:
The Django based web application is deployed on the Google App Engine and uses Google SQL for database transactions and Google Storage for storing images. The web application consists of a Django framework which connects to the frontend and backend of the application. The Google App engine gives the web application an endpoint once it is deployed. Using this endpoint, the client can access HTML pages and interact with the web application. The Google App engine is responsible for accepting such client requests, retrieving, or sending the necessary data from databases using Google Cloud SQL and retrieving.
When the volume of users increases the main bottlenecks of the system would be handling requests by the Google App Engine through load balancing, handling high volume database transactions by Google Cloud SQL.

d. The Google App engine is responsible for accepting client requests, retrieving, or sending the necessary data from databases using Google Cloud SQL. **Bottleneck** of our project is when the volume of users increases and when all of them trying to create events and food posts into database. The main **bottlenecks** of the system would be handling requests by the Google App Engine through load balancing, handling high volume database transactions by Google Cloud SQL.
Google Cloud SQL and Google App Engine are designed by Google to be elastic, reliable and auto scale in and out as required, the system will perform well when a high volume of users is using the system.

e. Solution:
when the students select a particular university, this web app shows all the available free food and events across the university. This web app also allows you to post the upcoming events happening on or around the selected campus.
Initially, if the employees do not have an account, they can create it. This data will be stored in Google cloud SQL. When the employees sign in, they can create a post on food/events going on in their university. The students can also sign up if they do not have an account. After the students sign in, they can search the food availability/ ongoing events and book for them. As soon as the student’s book, they get a booking id and an email with the

booking id number. The students can also subscribe to this university such that as soon as the employees post about food/events, the subscribed students will get the mail.

f. **Solution:**
As mentioned earlier we do not have any existing solutions for the food problem.

4. Testing and evaluation

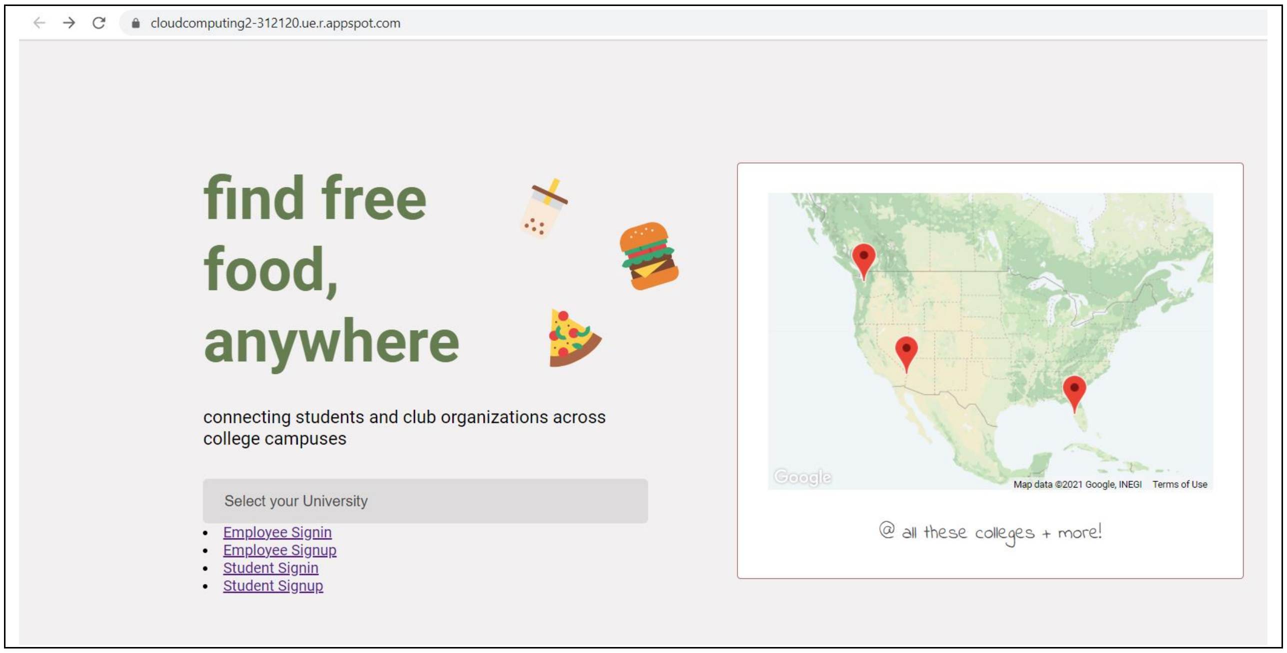
Application Testing Component- Testing Frontend, Testing Proper rendering of all the components and webpages This involved checking for all Frontend validations, if they are working correctly or not. Proper redirection and connection of all links Adaptive design to most common resolution (1920 X 1080)

Backend Testing - During this phase we tested each component of the backend using the Postman application to call each API individually using REST.

Integration Testing Scenario 1: Event post and food post end to end testing: Tested if all the posts in the postEvent and postFood table are populated in the database. Further, using Search food and Search event verified if we are able to fetch all the posts, from respective tables. On the frontend side, we ensured all the forms, their fields are displayed correctly. Correct display of all success, error and warning messages. We used the “Postman” application to invoke API on the backend and to verify if the response is correct. Initially we ran the application locally on our computer and used a locally hosted Postgressql database to verify if all the model’s operations are working properly as per our business logic. Then we installed google cloud-sql-proxy which we ran on our local computer. This cloud-sql-proxy was connected to the cloud-sql instance mapped to our project. During this phase we were still running the application locally. Once we ensured that the datapath and business logic is working as expected, we than moved to next step i.e to deploy the application on the App-Engine. We then performed all the end-to-end testing and made sure that all the components are working as expected by performing all the steps mentioned in Integration testing and Component testing.

Autoscaling of our web application which is deployed on Google App Engine is tested using Apache bench. Apache Bench (ab) is a load testing and benchmarking tool for Hypertext Transfer Protocol (HTTP) server. It can be run from command line and it is very simple to use. A quick load testing output can be obtained in just one minute. In our project using Apache Bench we hit 2000 requests, 16 instances are created (Scaling-Up).

a. App URL: https://cloudcomputing2-312120.ue.r.appspot.com/


cloudcomputing2-312120.ue.r.appspot.com

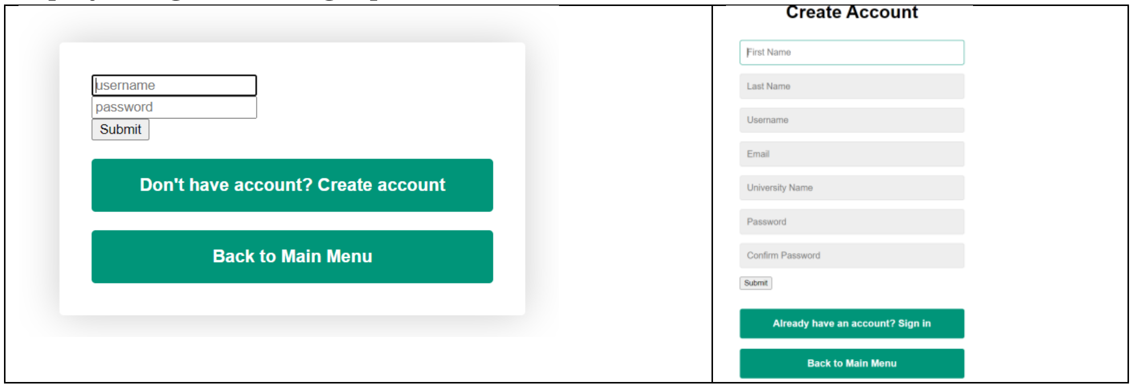
find free food, anywhere

connecting students and club organizations across college campuses

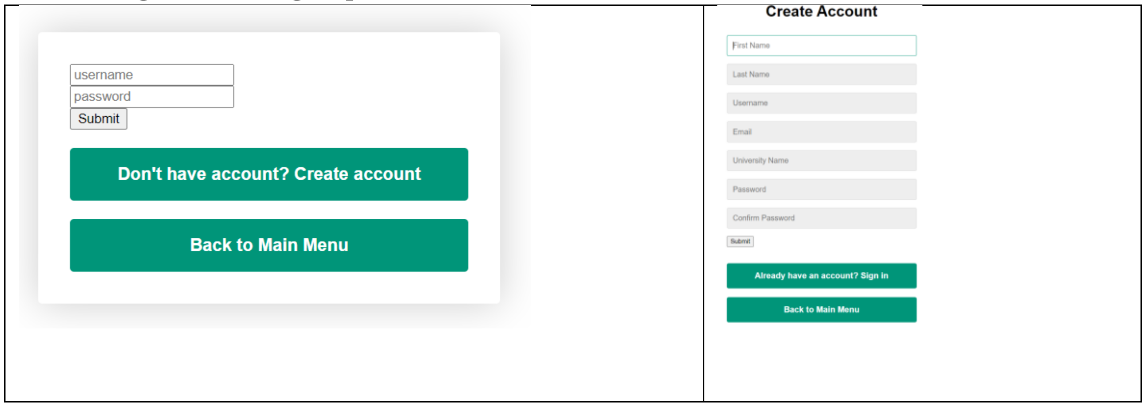
Select your University

- Employee Signin
- Employee Signup
- Student Signin
- Student Signup

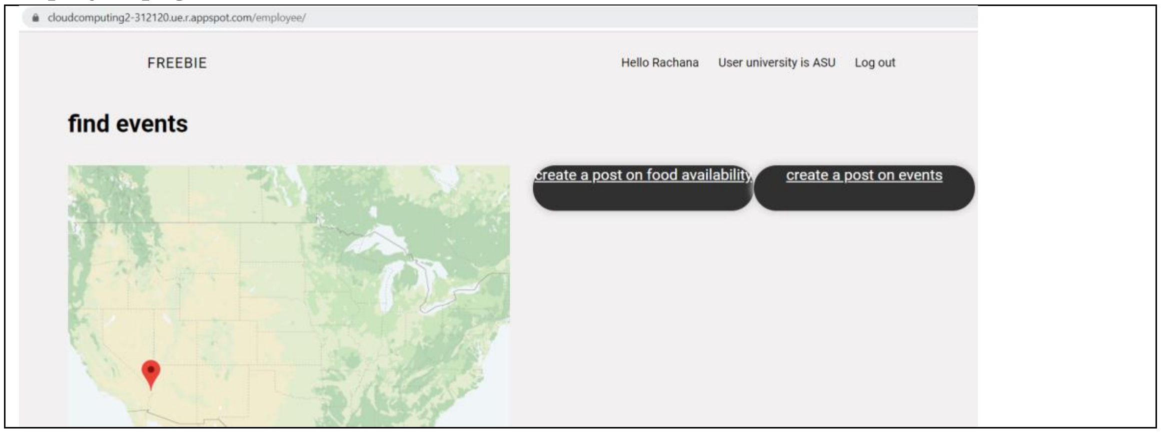
b. Employee Sign in and Signup:



c. Student Sign in and Sign up



d. Employee page:


cloudcomputing2-312120.ue.r.appspot.com/employee/

FREEBIE

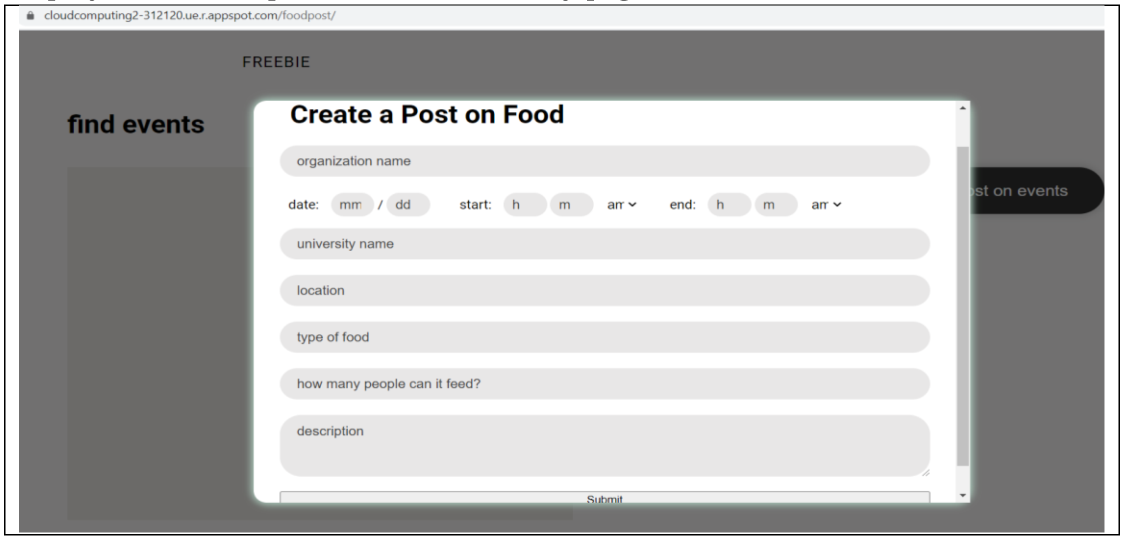
Hello Rachana User university is ASU Log out

find events

create a post on food availability

create a post on events

e. Employee - create a post on food availability page:


cloudcomputing2-312120.ue.r.appspot.com/foodpost/

FREEBIE

find events

Create a Post on Food

organization name

date: mm / dd start: h m am end: h m am

university name

location

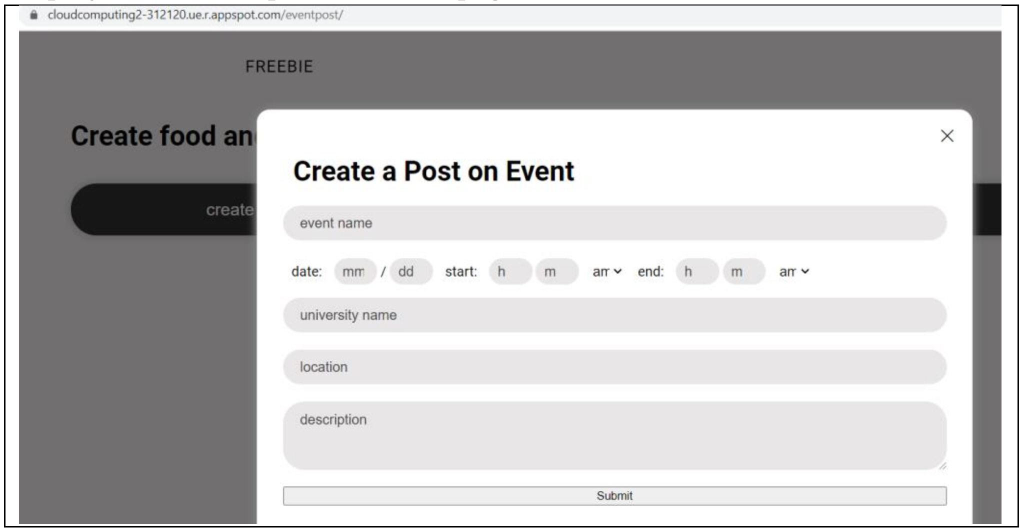
type of food

how many people can it feed?

description

Submit

f. Employee - create a post on events page:


cloudcomputing2-312120.ue.r.appspot.com/eventpost/

FREEBIE

Create food an

Create a Post on Event

event name

date: mm / dd start: h m am end: h m am

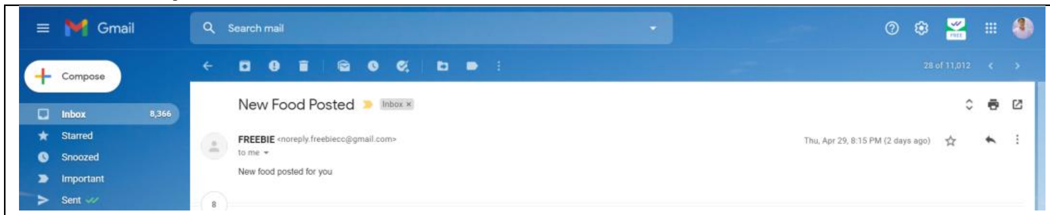
university name

location

description

Submit

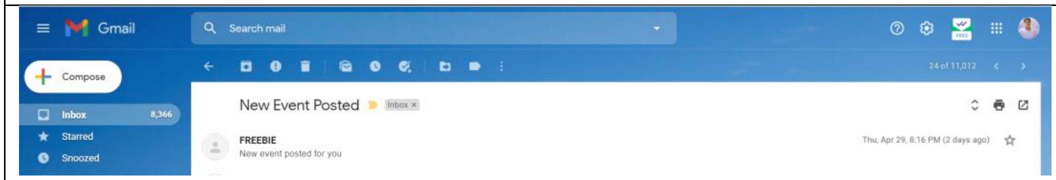
g. Whenever employees post about food availability or events, every subscribed student at that university will receive a mail notification:



New Food Posted — Inbox

FREEBIE <noreply.freebiecc@gmail.com>
to me
Thu, Apr 29, 8:15 PM (2 days ago)

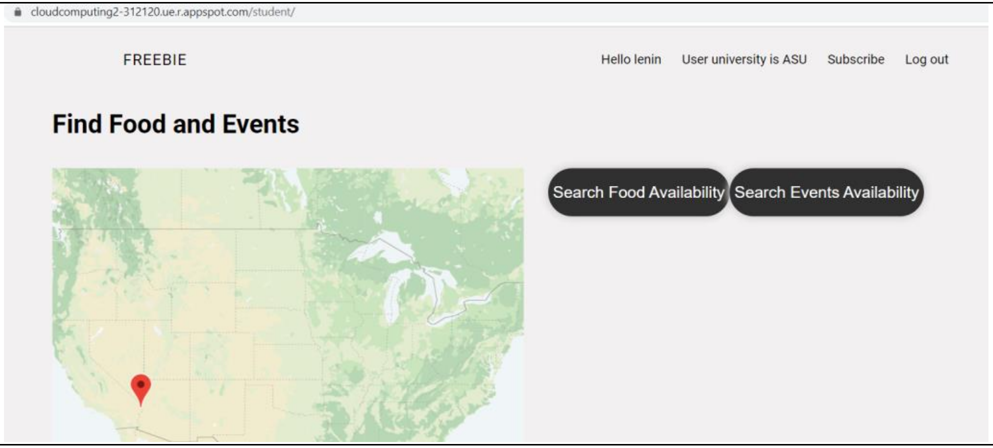
New food posted for you



New Event Posted — Inbox

FREEBIE
New event posted for you
Thu, Apr 29, 8:16 PM (2 days ago)

h. Student Page:

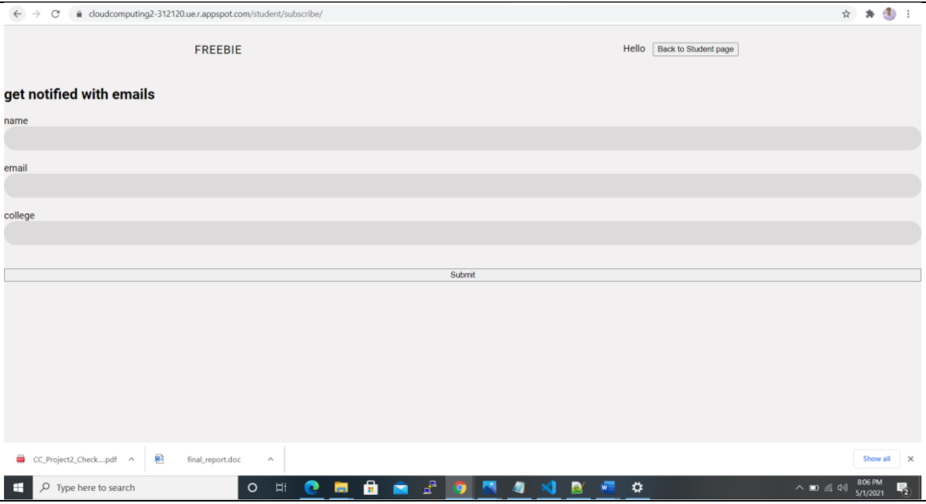


cloudcomputing2-312120.ue.r.appspot.com/student/

FREEBIE   Hello lenin   User university is ASU   Subscribe   Log out

Find Food and Events

i. Subscription page:



cloudcomputing2-312120.ue.r.appspot.com/student/subscribe/

FREEBIE   Hello [Back to Student page]

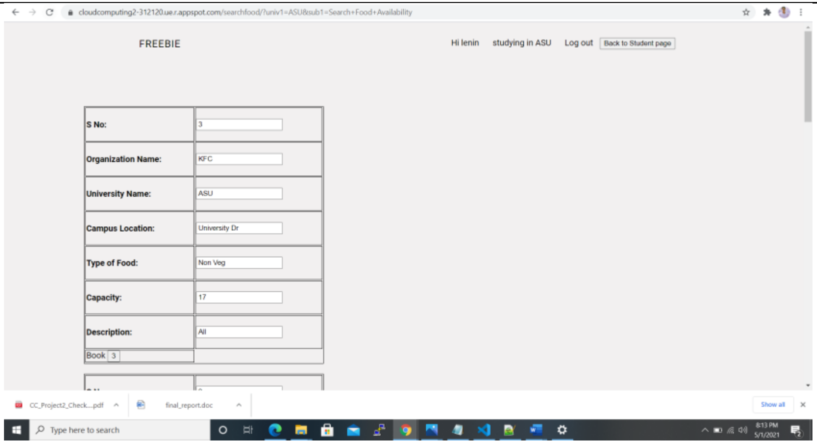
get notified with emails

name

email

college

j. Student – Search food availability page, generating a booking id and mail to the student:

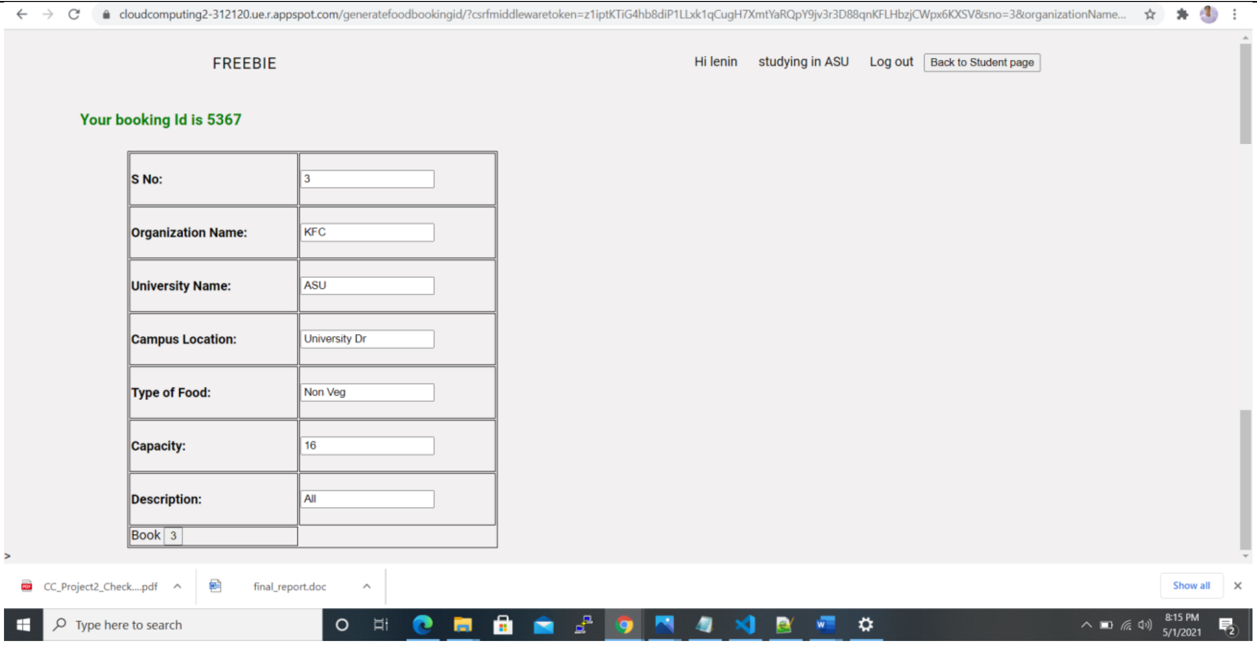


cloudcomputing2-312120.ue.r.appspot.com/searchfood/?univ1=ASU&sub1=Search+Food+Availability

FREEBIE

Hi lenin studying in ASU Log out Back to Student page

S No:	3
Organization Name:	KFC
University Name:	ASU
Campus Location:	University Dr
Type of Food:	Non Veg
Capacity:	17
Description:	All
Book 3	



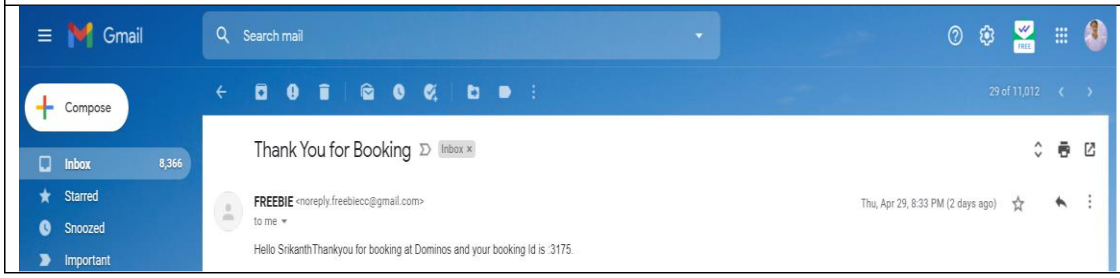
cloudcomputing2-312120.ue.r.appspot.com/generatefoodbookingid/?csrfmiddlewaretoken=z1iptKTiG4hb8diP1LLxk1qCugH7XmtYaRQpY9jv3r3D88qnKFLHbzjCWpx6KXSV&sno=3&organizationName...

FREEBIE

Hi lenin studying in ASU Log out Back to Student page

Your booking Id is 5367

S No:	3
Organization Name:	KFC
University Name:	ASU
Campus Location:	University Dr
Type of Food:	Non Veg
Capacity:	16
Description:	All
Book 3	



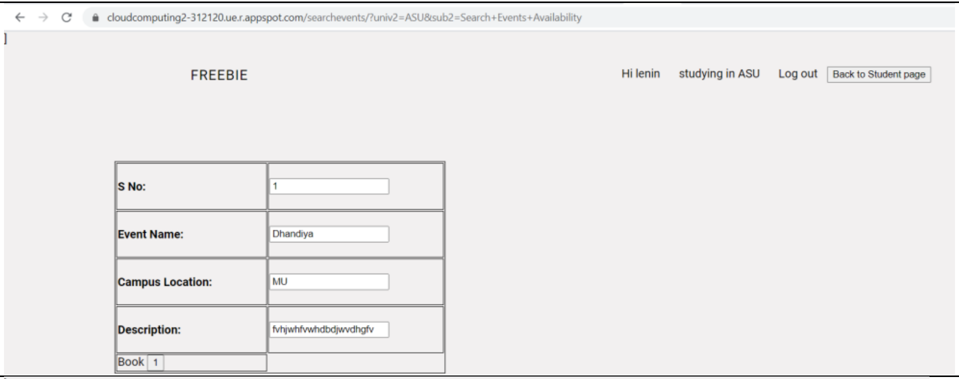
Thank You for Booking Inbox

FREEBIE <noreply.freebiecc@gmail.com> Thu, Apr 29, 8:33 PM (2 days ago)

to me

Hello SrikanthThankyou for booking at Dominos and your booking Id is :3175.

k. Student – Search events page, generating a booking id and mail to the student:

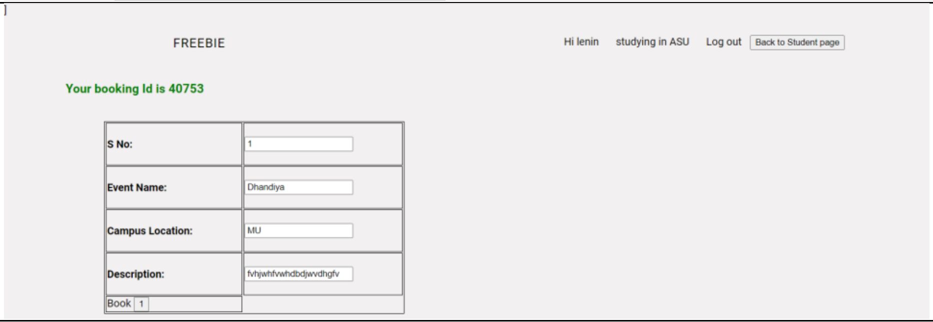


cloudcomputing2-312120.ue.r.appspot.com/searchevents/?univ2=ASU&sub2=Search+Events+Availability

FREEBIE

Hi lenin studying in ASU Log out Back to Student page

S No:	1
Event Name:	Dhandiya
Campus Location:	MU
Description:	fvhjwhfvwhdbdjwvdhgfv
Book 1	



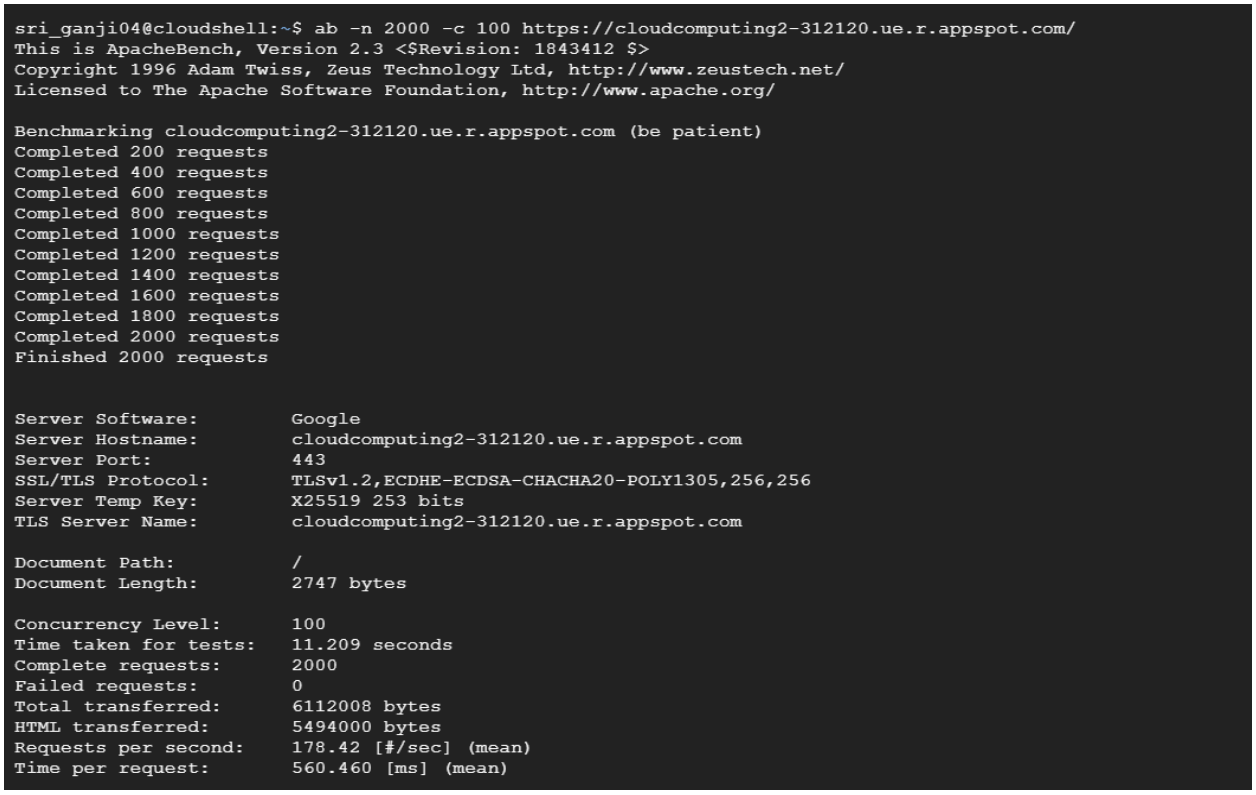
FREEBIE

Hi lenin studying in ASU Log out Back to Student page

Your booking Id is 40753

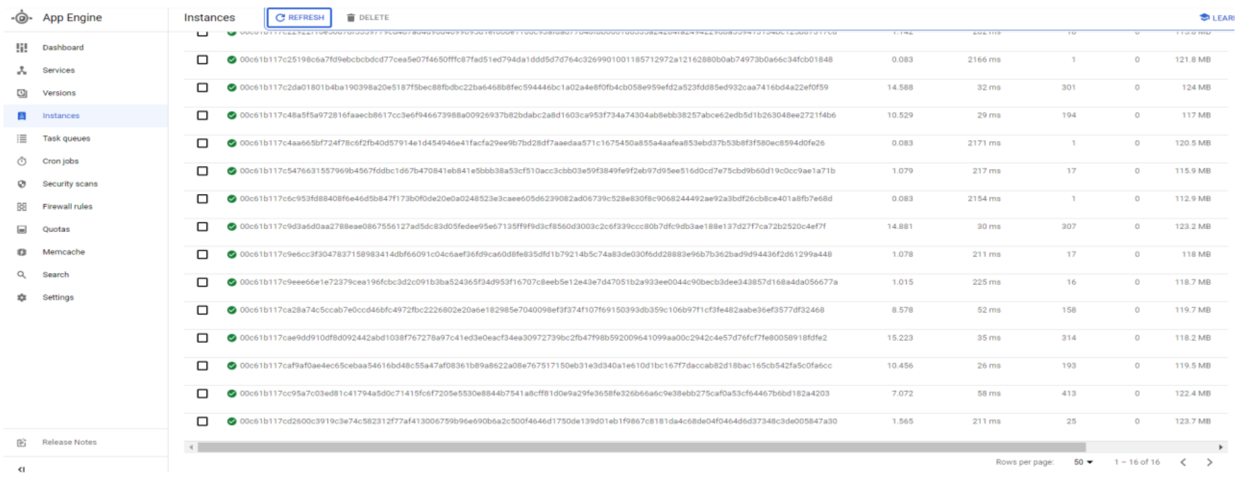
S No:	1
Event Name:	Dhandiya
Campus Location:	MU
Description:	fvhjwhfvwhdbdjwvdhgfv
Book 1	

l. Below is the Cloud shell editor. We hit the web page with 2000 requests



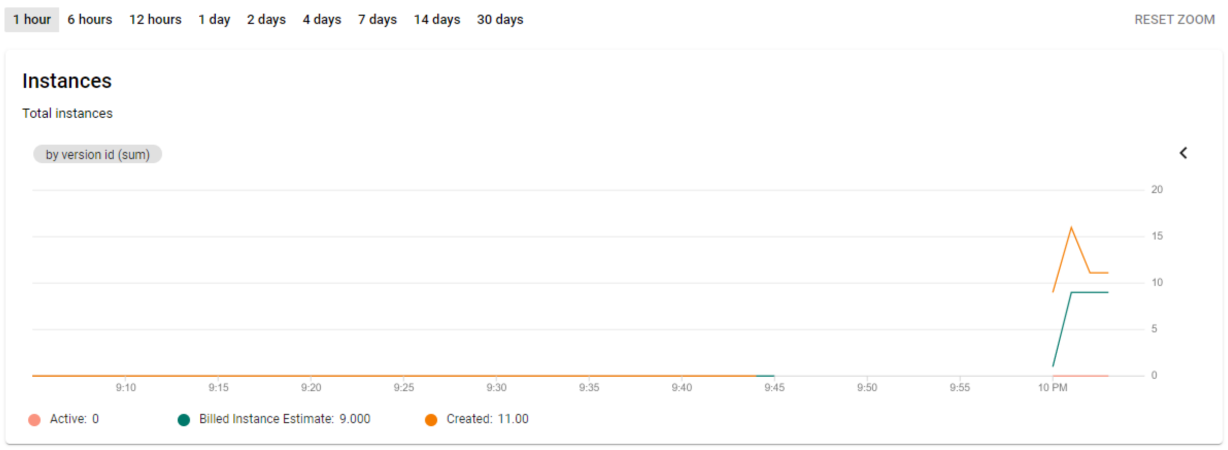
```
sri_ganji04@cloudshell:~$ ab -n 2000 -c 100 https://cloudcomputing2-312120.ue.r.appspot.com/
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking cloudcomputing2-312120.ue.r.appspot.com (be patient)
Completed 200 requests
Completed 400 requests
Completed 600 requests
Completed 800 requests
Completed 1000 requests
Completed 1200 requests
Completed 1400 requests
Completed 1600 requests
Completed 1800 requests
Completed 2000 requests
Finished 2000 requests


Server Software:        Google
Server Hostname:        cloudcomputing2-312120.ue.r.appspot.com
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-ECDSA-CHACHA20-POLY1305,256,256
Server Temp Key:        X25519 253 bits
TLS Server Name:        cloudcomputing2-312120.ue.r.appspot.com

Document Path:          /
Document Length:        2747 bytes

Concurrency Level:      100
Time taken for tests:   11.209 seconds
Complete requests:      2000
Failed requests:        0
Total transferred:      6112008 bytes
HTML transferred:       5494000 bytes
Requests per second:    178.42 [#/sec] (mean)
Time per request:       560.460 [ms] (mean)
```

m. Below is the App Engine dashboard, here we can see that for 2000 requests 16 instances are created.



[illegible]	1.142	202 ms	16	0	[illegible]
00c61b117c25198c6a7fd9ebcbcbdcd77cea5e07f4650fffc87fad51ed794da1ddd5d7d764c3269901001185712972a12162880b0ab74973b0a66c34fcb01848	0.083	2166 ms	1	0	121.8 MB
00c61b117c2da01801b4ba190398a20e5187f5bec88fbdbc22ba6468b8fec594446bc1a02a4e8f0fb4cb058e959efd2a523fdd85ed932caa7416bd4a22ef0f59	14.588	32 ms	301	0	124 MB
00c61b117c48a5f5a972816faaecb8617cc3e6f946673988a00926937b82bdabc2a8d1603ca953f734a74304ab8ebb38257abce62edb5d1b263048ee2721f4b6	10.529	29 ms	194	0	117 MB
00c61b117c4aa665bf724f78c6f2fb40d57914e1d454946e41facfa29ee9b7bd28df7aaedaa571c1675450a855a4aafea853ebd37b53b8f3f580ec8594d0fe26	0.083	2171 ms	1	0	120.5 MB
00c61b117c5476631557969b4567fddbc1d67b470841eb841e5bbb38a53cf510acc3cbb03e59f3849fe9f2eb97d95ee516d0cd7e75cbd9b60d19c0cc9ae1a71b	1.079	217 ms	17	0	115.9 MB
00c61b117c6c953fd88408f6e46d5b847f173b0f0de20e0a0248523e3caee605d6239082ad06739c528e830f8c9068244492ae92a3bdf26cb8ce401a8fb7e68d	0.083	2154 ms	1	0	112.9 MB
00c61b117c9d3a6d0aa2788eae0867556127ad5dc83d05fedee95e67135ff9f9d3cf8560d3003c2c6f339ccc80b7dfc9db3ae188e137d27f7ca72b2520c4ef7f	14.881	30 ms	307	0	123.2 MB
00c61b117c9e6cc3f3047837158983414dbf66091c04c6aef36fd9ca60d8fe835dfd1b79214b5c74a83de030f6dd28883e96b7b362bad9d94436f2d61299a448	1.078	211 ms	17	0	118 MB
00c61b117c9eee66e1e72379cea196fcbc3d2c091b3ba524365f34d953f16707c8eeb5e12e43e7d47051b2a933ee0044c90becb3dee343857d168a4da056677a	1.015	225 ms	16	0	118.7 MB
00c61b117ca28a74c5ccab7e0ccd46bfc4972fbc2226802e20a6e182985e7040098ef3f374f107f69150393db359c106b97f1cf3fe482aabe36ef3577df32468	8.578	52 ms	158	0	119.7 MB
00c61b117cae9dd910df8d092442abd1038f767278a97c41ed3e0eacf34ea30972739bc2fb47f98b592009641099aa00c2942c4e57d76fcf7fe80058918fdfe2	15.223	35 ms	314	0	118.2 MB
00c61b117caf9af0ae4ec65cebaa54616bd48c55a47af08361b89a8622a08e767517150eb31e3d340a1e610d1bc167f7daccab82d18bac165cb542fa5c0fa6cc	10.456	26 ms	193	0	119.5 MB
00c61b117cc95a7c03ed81c41794a5d0c71415fc6f7205e5530e8844b7541a8cff81d0e9a29fe3658fe326b66a6c9e38ebb275caf0a53cf64467b6bd182a4203	7.072	58 ms	413	0	122.4 MB
00c61b117cd2600c3919c3e74c582312f77af413006759b96e690b6a2c500f4646d1750de139d01eb1f9867c8181da4c68de04f0464d6d37348c3de005847a30	1.565	211 ms	25	0	123.7 MB

Rows per page: 50 1 – 16 of 16



5. Code

a) Frontend:

- index1.html -> It is the home page for the application. Its corresponding script is written in indexScript2.js.
- empCreate.html -> It contains employee signup form.
- empLogin.html -> It contains the employee login form.
- employeePage.html -> As soon as the valid employee credentials are entered, it redirects student to this page. It contains the create food post and create event post buttons. Its corresponding script is written in employeeScript.js.
- stdCreate.html -> It contains student signup form.
- stdLogin.html -> It contains student login form.
- studentPage.html -> As soon as the valid student credentials are entered, it redirects employee to this page. It contains the create food post and create event post buttons. Its corresponding script is written in studentScript.js.

- foodPost.html -> on clicking the create food post button it redirects to this page. It contains the form to create food post. Its corresponding script is written in foodPost.js.
- eventPost.html -> on clicking the create event post button it redirects to this page. It contains the form to create event post. Its corresponding script is written in eventPost.js.
- searchFood.html -> on clicking the search food button in studentPage.html, it gets redirected to this page. It contains the forms that displays the available food . We can register for the food take away by clicking the book button. Its corresponding script is written in searchFood.js.
- searchEvents.html -> on clicking the search events button in studentPage.html, it gets redirected to this page. It contains the forms that displays the available events. We can register for the events by clicking the book button. Its corresponding script is written in searchEvents.js.
- mail.html -> It contains the template for body of mail.
- Subscribe.html -> when ever a student clicks on subscribe button it renders this page. It contains the subscription form. Its corresponding script is written in subscribe.js.

b) Backend:

- models.py -> In this we created database layout, with additional metadata. It contains methods defined to create tables in database like event post and food post tables.
- urls.py -> It contains all url patterns that are to be directed from the home page to the methods in views.py.
- views.py -> It contains the methods which define the dynamic behavior of all the html pages.

Methods in the views.py are:

- empsignin(): validates the employee and if correct redirects to the employeePage.html, otherwise shows the message as invalid credentials.
- Empcreate(): creates a new employee credentials in the employee table and redirects to the employee login page.
- Stdsignin(): validates the student and if correct redirects to the studentPage.html, otherwise shows the message as invalid credentials.
- stdcreate(): creates a new student credentials in the student table and redirects to the student login page.
- Emppage(): It redirects to employeepage.html
- Stdpage(): It redirects to studentpage.html
- Foodpost(): It gets the foodpost data from the foodpost.html and inserts into the postFood1 table. It sends mail to all subscribers saying new food is posted.
- Eventpost():It gets the eventpost data from the eventpost.html and inserts into the postEvent1 table. It sends mail to all subscribers saying new event is posted.
- Searchevents(): It fetches the university specific event posts and displays on the search events page.
- Searchfood():It fetches the university specific food posts and displays on the search food page.
- generateFoodBookingId(): As soon as the students books a food , food capacity reduces by 1 and a booking ID is generated and the same is mailed to their gmail account.

- generateEventBookingId(): As soon as the students books an event, a booking ID is generated and the same is mailed to their gmail account.
- subscribeUser(): when a student selects the subscribe button it takes the details and inserts into subscribe table of the database.
- send_mail(): It takes subject , body, from and to addresess of mail in order to send mails to either subscribers or students.

c) Settings.py: This file consists of all the environment variables that are used throughout the execution of the application.

d) Urls.py: This file consists of all the urls, that expose the API for specific business logic.

e) app.yaml: This file acts as a deployment descriptor of a specific service version. This is essential for deploying apps on the App-engine.

f) Main.py: This file imports the WSGI-compatible object of your Django app, application from ccapp/wsgi.py and renames it app so it is discoverable by App Engine without additional configuration.

Installation Steps In order to deploy the application on the app engine please follow below mentioned steps:

Install python3.7.

Install gcloud sdk for your operating system.

Create a project on the google cloud. Under this project, make an instance of Cloud sql, and make changes in the settings.py for the variable Database, so that your application connects to the right database when deployed.

Setup gcloud for your google account and select your newly created project.

Go to the root directory of the django application, where you can find manage.py.

Command to deploy the application on the App-Engine:

```
$gcloud app deploy.
```

```
$gcloud app browse.
```

6. Conclusions:

- This project helped us realize how easy it is to take a simple day to day idea, implement and realize it and deploy it on the cloud so that it can be used publicly by users accessing the application in high volumes. Using Google App engine removes the headache of optimally handling the cloud infrastructure and autoscaling of the application. Thus, the focus of developers can be on developing their application, unlike the previous project where optimal auto scaling, resource utilization and reduced latency was a big concern.

- **Future Scope**: We want to develop a mobile application and implement pop up notifications if the student passes by any college where food and events are available.

7. References:

- QuickStart for Python 3 in the App Engine Standard Environment
- Writing your first app in Django tutorial
  https://docs.djangoproject.com/en/3.0/intro/tutorial01/