Project Information

Title: CaterGeek

Members:

1. **Akshay – akshayjain1996**
2. **Sneh – snehpatel3**
3. **Priyen – c5patelp**
4. **Siddharth – Sidd0107**

Description:

CaterGeek is a website where users' food requirements for events or celebrations, or anything are met. Users who have a need for food can sign up and check out profiles' of caterers. Different caterers are capable of different things and once a user finds a good caterer, they can make an order related to their needs. A user is able to sort through the listings by rating of the caterer, specialties, and more.

A User can also activate a caterer profile after they sign up, should they want to be a caterer. Once a caterer profile is activated, they can list their specialties, they will have the opportunity to pickup/accept/refuse/complete orders. We leave the financial details up to the caterer/user.

Users can leave reviews of caterers and caterers maintain a satisfaction rating and list of reviews.

Detailed Design:

In order to run the app, please read readme.pdf.

Software Framework design decision: We used Angular JS and Bootstrap frameworks since a majority of the team was comfortable with it and it made it easier to implement certain aspects in a clean manner including routing, creating requests and dealing with the server.

Hierarchy Design: CaterGeek is made up of two types of users. One type of user is a caterer, the other is a regular user. Caterers can view, accept or reject orders that were placed and view reviews that were written about them. Additionally, they can also add interesting details to their profile such as price range, speciality cuisine, address etc. Regular users wishing to place orders can then view details about these caterers, place order requests, write reviews on caterers and also view the caterers with the highest ratings(Which are suggested to them). They can also

Page Specific features/Design:

Login Page: Upon entering the URL, the user is redirected to a sign in page. He/She can either sign in or register a new account. Based on our app design, all members must sign up before they can use the app. This design decision was taken to make it easier to keep track of orders and users. Both frontend and backend measures are taken in order to validate the fields. If the user has an account and sign's in, he/she is redirected to their dashboard.

Register Page: If the user does not have an id and clicks on register, he is redirected to this page from the login page. Here the user fills up some basic details such as Display name, email id and password. These are once again validated and then the user is registered. Initially, every user is registered as a basic user.

User Dashboard: Every user registering and regular user signing in is taken to a user dashboard. Here they can click on profile to update their profile. They can click on logout to logout. Or they can click on Caterer dashboard to make themselves a caterer. If they make themselves a caterer, they are automatically rerouted to a caterer accounts dashboard. Apart from that, users and caterers can view a list of highest ranked caterers (Based on user ratings) which are the caterers recommended by the website and also another list of all other caterers(they can also click on a button to view each caterer's details). Here they can search for specific caterers as well by typing in their name or cuisine type that they desire or even specialities. Here they can also view the status of their orders.

Caterer Dashboard: Over here, every logged in caterer and newly registered caterers can view their new order requests and accept or decline them. Apart from that, they can also view pending orders that they have accepted and also completed orders that they have executed.

Update User: If a regular user clicks on update, he/she can update specific fields of their account. This includes display name, favourite dish and their password.

Update Caterer: If a caterer clicks on update, he/she can update their price range, address, password, display name, description and also add a cuisine type that they might have learned. Users can then view this when they click on view besides a regular user.

View Caterer: This page allows users and caterers to view caterer specific details. From here, they can also leave a review and ratings. Apart from that, orders are also placed from here by clicking on the place order button. Automatically the user and order details are sent to the caterer which can either be accepted or rejected.

The reason the design was planned in this was because it makes things simple. It almost follows the singleton design pattern where each page is given a primary responsibility, which is either to update, or to place an order or to allow a user to sign in(Apart from the usual displaying responsibility of every webpage). This made the coding process much more streamlined. Apart from that, on the backend, controllers were used to manage data processing and communication with the server. Each html file had its own controller in the client Controller file. These decisions lead to a quick and efficient code base.

Security Testing:

Please See https://piazza.com/class/iea8t9q0fbj1d0?cid=404

- **Our focus is on #2 and #5**
- Preventing DOM-based XSS
  - We do not include any user input in JS generated HTML.

  - Our user input is processed as text and not html, preventing dom-based XSS
  - We also use the httpOnly: true cookie properly to make XSS attacks harder
- Protection against DDOS
  - We are using a host called Heroku which provides protection against DDOS using the following:
    - TCP Syn cookies and connection rate limiting
    - Using multiple backbone connections
    - internal bandwidth capacity that is more than the internet carrier supplied bandwidth
  - They provide strong infrastructure against these attacks and additionally enable advanced mitigation techniques when needed.
  - **Evidence: https://www.heroku.com/policy/security**
- Attempting to use RESTApi without appropriate permissions
  - All clients are forbidden and this data is only accessible through the UI(calls from hidden angular code)
- Attempting to access pages without being logged in or non-existent pages
  - A blank screen or forbidden message is shown

Performance Optimizations:

We ran heavy background tests to check the performance of our web application. Initially it turned out that these tests were causing the performance to deteriorate. After a lot of analysis we noticed that it was because we were overburdening the server with work. Then we had to modify our design so that we process more data on the user side and less on the client side. A lot of the heavy validation checks that were only required on the client side or other calculations that were required were all done on the client side. Apart from that, another aspect that was severely reducing the performance was getting too much data from the database via the server. Later on, instead of sending a huge chunk of the database, we filtered out specific field that were required and sent them back to the client. These two major steps improved our app’s performance significantly.

Video Demo:

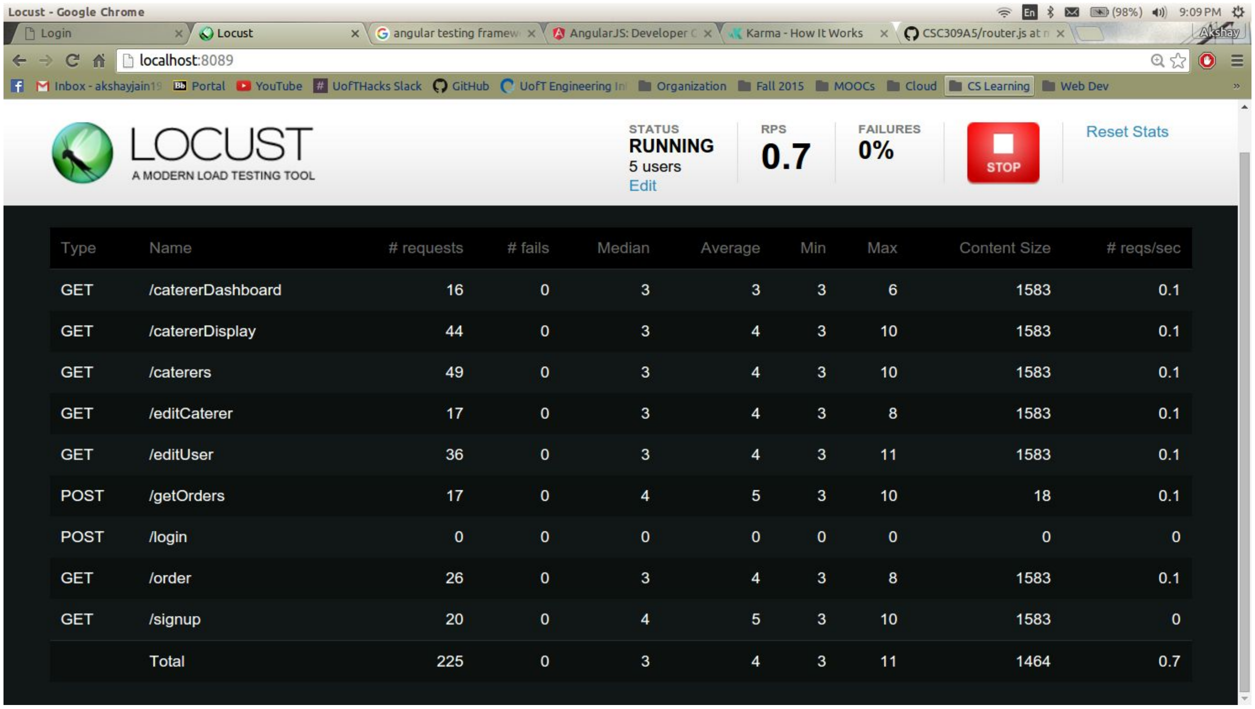
http://screencast.com/t/hhlbmsUXC72g

Heroku:

https://immense-springs-8164.herokuapp.com/

Testing:

We use a testing framework Locust (http://locust.io/) to throughly test our app. We test all our routes to make sure they are working at all times. The testing code is in the file locust.py. Here is a screenshot of the test we ran:



STATUS RUNNING 5 users

RPS 0.7

FAILURES 0%

Type	Name	# requests	# fails	Median	Average	Min	Max	Content Size	# reqs/sec
GET	/catererDashboard	16	0	3	3	3	6	1583	0.1
GET	/catererDisplay	44	0	3	4	3	10	1583	0.1
GET	/caterers	49	0	3	4	3	10	1583	0.1
GET	/editCaterer	17	0	3	4	3	8	1583	0.1
GET	/editUser	36	0	3	4	3	11	1583	0.1
POST	/getOrders	17	0	4	5	3	10	18	0.1
POST	/login	0	0	0	0	0	0	0	0
GET	/order	26	0	3	4	3	8	1583	0.1
GET	/signup	20	0	4	5	3	10	1583	0
	Total	225	0	3	4	3	11	1464	0.7