Group 6 - Holler Final Project

When we were thinking of Ideas, Cleon was thinking about doing a food app. And suggested Ian a delivery app. Then we were discussing whether this would be a global or a local app. We as a group decided it would be a global app.

Why this app?

- Help reduce viral spread by keeping more people indoors and hence will save many lives
- Can be used with businesses that provide necessary services (such as medicine, food, groceries)
- Localized delivery, familiarity with the community and surrounding areas
- Repeatable model in any country or city
- Delivery services can sign up to receive delivery requests

Name ideas:

- Hermes
- Right on Time
- Haste
- Rush
- **Holler (selected name)**
- Jot
- Hop

Features

- Ian
    - Admin UI
    - Package UI
    - Package Management
- Cleon
    - Delivery Service UI
    - Delivery Service Management
    - Messaging and Notifications
- Bishajit
    - Recipient UI
    - Search engine
    - Route Management
- Ibrahima
    - Billing/Invoicing UI: for payments
    - Registration/Sign Up UI: for authentication

- Geolocalization: for nearby available services; tracking ETA

Microservices Architecture Selected

- Scalability- We are trying to become a global product
- Adding new features to individual services
- Flexibility in terms of improvement to individual microservices
- Easier to troubleshoot and distributed maintenance

Setting up GIT

1. Git repository created by manager
2. Then each member went to their home directory and enter ~/.ssh to go to the .ssh directory
3. Members then generated new ssh public and private keys using ssh-keygen -t rsa -b 4096 -C “email@email.com" (put in your email instead, erase the quotation mark)
    a. Members were then prompted to enter the name of the ssh key (such as id_rsa). Some members created new ssh key names as they had previously used this name.
    b. Members were then prompted to enter a passphrase which we opted not to do.
4. Public and private ssh keys were then generated.
5. Members then used exec ssh-agent bash to start a new instance of the bash shell and replace the older one.
6. Members then started the ssh-agent in the background using $ eval "$(ssh-agent -s)"
7. Members then added the private key to the SSH agent using $ ssh-add ~/.ssh/ssh_key_name
8. Each member then gave the public key to the group manager by going into the .ssh directory and opening up the ssh key using cat id_rsa.pub, then copying the key and pasting it in our Slack channel.
9. Manager added each group member to GitHub repository through invitation to collaborate.
10. All members cloned the repository in their home or desktop directory using git clone git@github.com:Bishajit/Holler.git

11. Except the manager, each member set our config file to include our user name and user email with:
    a. git config --global user.name “name”
    b. git config --global user.email “email”
12. Members viewed config file to verify change using git config --list
13. Members changed the branch name to main using git branch -M main
14. Members then set the origin upstream branch with git branch --set-upstream-to=origin/main main
15. Members created sample files with different names for each member with touch README-name.Then we added files to the staging environment with git add .
16. Then we committed the files to the origin/main with git commit -m “commit message”
17. Members then pulled remote commits to the remote/origin using git fetch --all
18. Members then check the commit statuses of local and remote repositories using:
    a. git log
    b. git log origin/main
19. Members then merged with the remote files with our local repository with git merge
    a. Some members got this error fatal: refusing to merge unrelated histories
    b. Those members used git merge --allow-unrelated-histories to bypass this error
20. Members then checked to ensure the remote files were downloaded to our local repository using ls
21. Then members were able to push local files to remote repositories using git push
22. Then we checked GitHub to ensure our local files were uploaded to the remote repository and found that they were.
23. One member then created new branch for features using git branch features
24. One member then listed all branches to ensure it was create using git branch
25. Member then switched to features branch using git checkout features
26. Member then pushed new branch to remote repository and setting upstream branch at the same time using git push --set-upstream origin features
27. Remaining members then ensured they had the features branch using git fetch --all
28. We deleted all our sample/test files using rm command
29. Then git push
30. After that git commit -m “deleted all the sample files”
31. Added our individual feature filles. Each member repeated steps from 32 to 37.

32. git add .
33. git commit -m “ “
34. git fetch
35. git merge
36. git push
37. ls to make sure the files are there in feature branch
38. We created a pdf of our google document and made it a pdf file. Used wget to retrieve the file from our remote server to our main branch.