Project Summary

In this project we implemented an online version of the “Privilege Walk” activity. The privilege walk is an activity used to highlight the differences between people in a group to help everyone become more aware of their different perspectives. In this activity, participants usually stand in a straight line across a room and respond to statements about privileges by taking steps forwards or backwards. At the end of the activity everyone is able to see each other’s positions based on their “privilege”. The biggest problem with this activity is that it makes participants feel uncomfortable. Especially participants who standout by being too far ahead or behind when compared to others. Because participants feel uncomfortable, this can discourage participants from being honest out of fear of standing out. In addition, this activity has to be done in an open space and everyone has to be in the same physical solution.

The solution we built for our client, Dr. Shawna Thomas, is an online tool that facilitates doing the privilege walk activity. This tool anonymizes participants so they don’t feel singled out while providing a host with anonymized statistics. Since the activity is online participants can participate remotely from any location with a good internet connection.

Team Roles

Product Owner & Developer - Joseph Nyangwechi
Scrum Master & Developer - Sindhuja Reddy Kamidi
Developer - Suraj Shamsundar Jain
Developer - Sankrandan Loke

User Stories implemented.

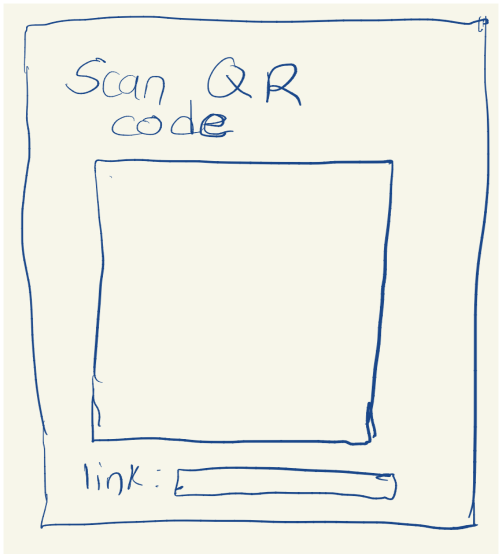
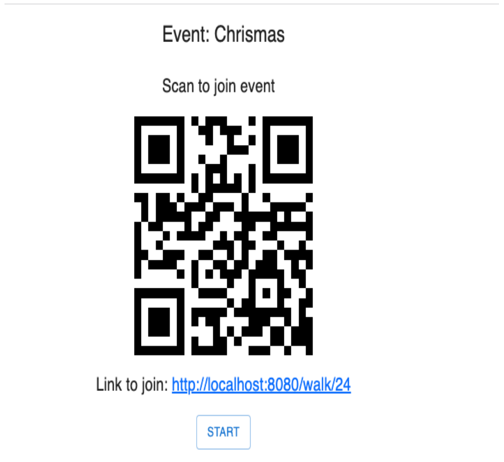
Feature short description	User story description	Points Allocated(contains sub tasks for each member)	Status	Change to User Story
Sign up and Login	As a privilege walk host, I want to be able to sign up and log in	3(8 chores)	Completed	
Anonymize participants while	As a privilege walk participant,	2(8 chores)	Completed	

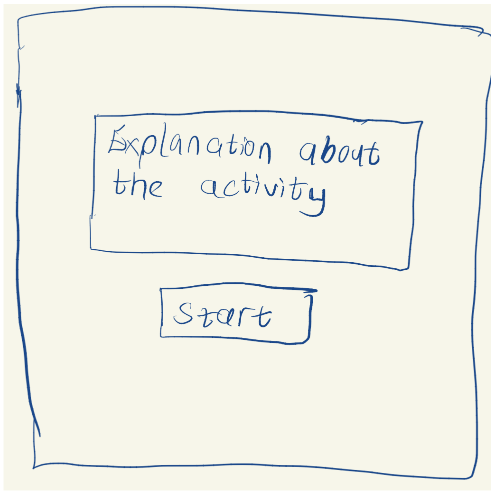
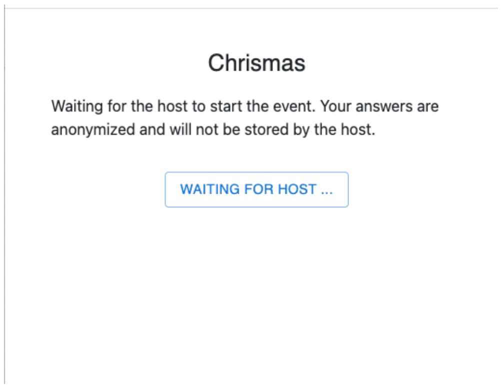
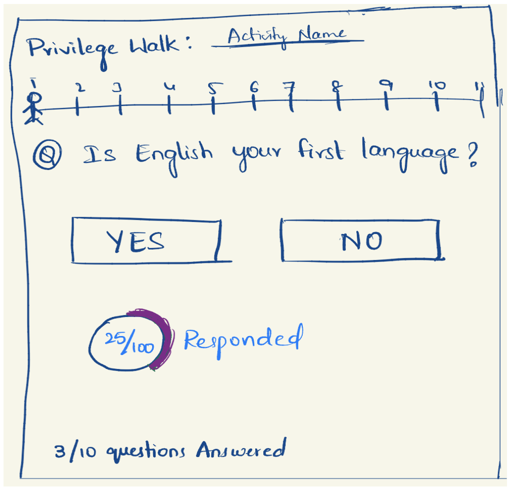
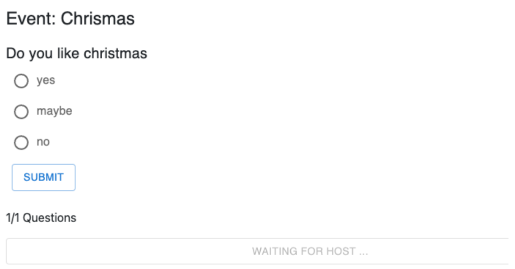
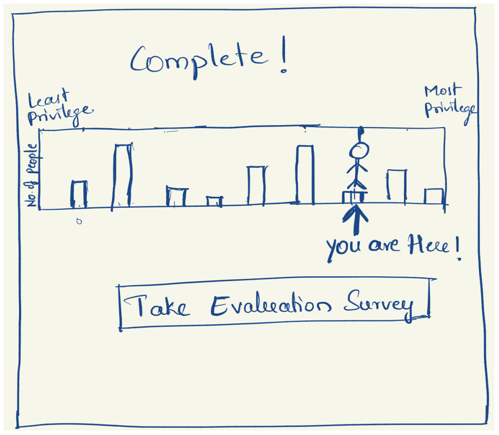
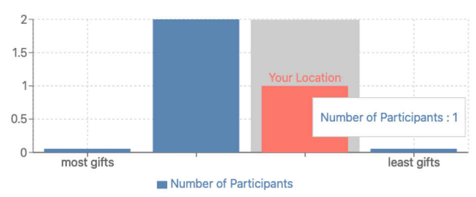
doing the privilege walk	So that I don't feel singled out while doing the privilege walk, I want my privilege walk answers to not be visible to other people while I'm doing the privilege Walk.			
Event short description and option to join	As a privilege walk participant So that i understand the event. I want to see a short description of the event before I join the event.	2(8 chores)	Compl eted	
Display the questions	As a privilege walk participant So that i can join the event and participate I want to see the question description and the options for answers when the event is live.	3(8 chores)	Compl eted	
Show participants anonymized privilege walk results	As a privilege walk participant So that I know how long I need to wait for the next question. I want to see the number of participants yet to answer the question as an indication of waiting time to get to the next question.	3(8 chores)	Compl eted	
Show the results of privilege walk after the event	As a privilege walk participant So that I know the results and my position after the event. I want to see my position among the participants and statistics of my privileges.	3(8 chores)	Compl eted	
Create new events	As a host of a privilege walk event So that my event attendees can participate in it. I want to create an event on the hosts dashboard and have it stored in the database.	3(8 chores)	Compl eted	

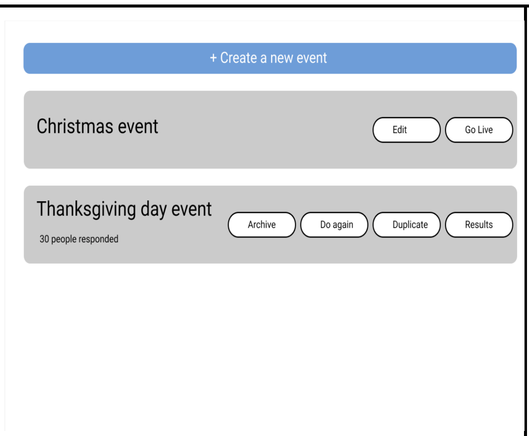
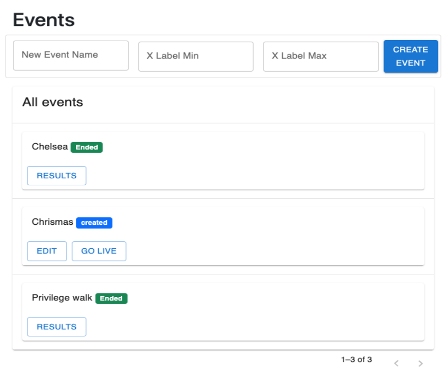
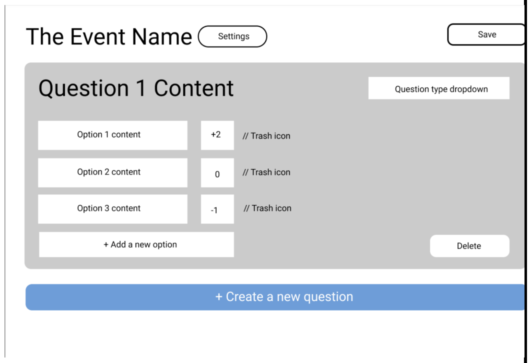
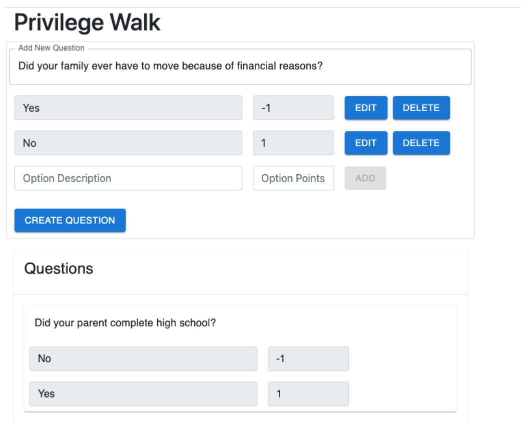
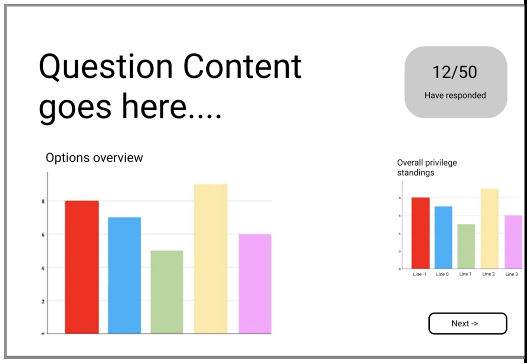
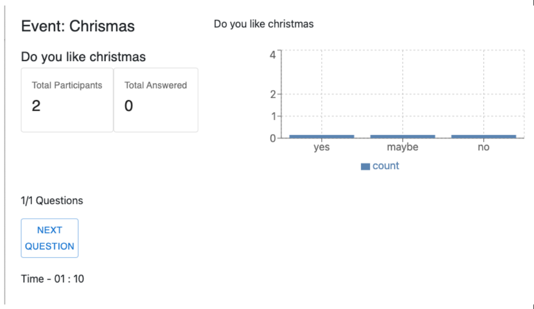
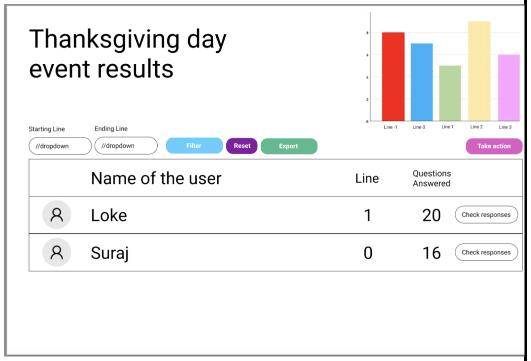
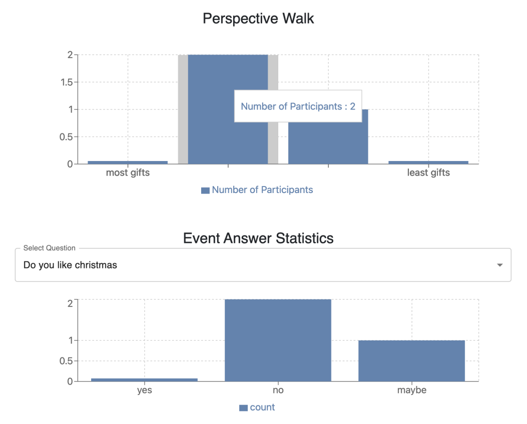
Add and customize questions in the events created	As a host of a privilege walk event So that my event attendees can answer them. I want to be able to add, remove, and edit the content of the questions using the host-side dashboard.	3(8 chores)	Compl eted	
Add points to the answer options	As a host of a privilege walk event So that attendees can move forward or backward in the line. I want to be able to set points to each of the options in the questions that determine the movement of the users choosing that option as an answer.	3(8 chores)	Compl eted	
Host the event live	As a host of a privilege walk event So that the attendees can connect to it. I want the attendees to be able to scan the QR code, join the event, and participate in the anonymous privilege walk. I want to have a link below the QR code, so that if QR code doesn't work i can use link to join event	2(8 chores)	Compl eted	
Release next question	As a host of privilege walk event I want the control to release the next question based on number of participants responded to the previous question	2(8 chores)	Compl eted	
View the live answers and analytics during the event	As a host of a privilege walk event So that I know the new attendees positions based on their answers I want to view a graph showing the number of attendees on each line	3(8 chores)	Compl eted	

View the results of the event	As a host of a privilege walk event So that I gain insights about the diversity of the participants in the activity I want to view the final results of the activity in the form of animation and charts	3(8 chores)	Completed	
Export results of the event	As a host of a privilege walk event So that I can access the results of the activity even after it ends I want to be able to download data in the form of charts and excel		NOT Completed	Not required anymore

lo-fi UI mockups/storyboards

User Story	Lofi-mockups	Screen shots
1.Host the event live		 Event: Chrismas Scan to join event Link to join: http://localhost:8080/walk/24

1.Event short description and option to join		Chrismas Waiting for the host to start the event. Your answers are anonymized and will not be stored by the host.
1.Anonymize participants while doing the privilege walk. 2.Display the questions. 3.Show participants anonymized privilege walk results.	 Privilege Walk : Activity Name 1 2 3 4 5 6 7 8 9 10 11 Q Is English your first language? YES NO 25/100 Responded 3/10 questions Answered	Event: Chrismas Do you like christmas ○ yes ○ maybe ○ no 1/1 Questions
1.Show the results of privilege walk after the event	 Complete! Least Privilege Most Privilege No. of people you are Here! Take Evaluation Survey	Perspective Walk 

1.Create new events		
1.Add and customize questions in the events created 2.Add points to the answer options		
1.View the live answers and analytics during the event		
1.View the results of the event		

Development Process (Scrum iterations)

Note: We count the number of chores for each user story we completed to get the total points completed for that iteration

Iteration 0:

Accomplishments:

1. Requirements gathering - client meetings
2. We came up with UI Lofi-diagrams for host portal and participant portal
3. Presented Lofi-diagrams to the client and collected feedback.
4. Documented user stories.

Points completed in this iteration: 8

Iteration 1:

Accomplishments:

1. Decided on the technology stack to be used for implementation of project.
2. After research and based on the client requirements, we decided to go for Django and React.
3. Initial setup of the project
    a. Created backend github repo
    b. Created frontend github repo
    c. Established connection for backend and frontend.
4. Completed user signup/login for both frontend and backend.
5. Designed database models
    a. https://lucid.app/lucidchart/901b0e43-df07-48df-bdc9-1ae8d9c21f7c/edit?invitationId=inv_9b5c53ff-65f9-4977-80c7-e0b6e4731c41&page=J960fNmy6zCK#

Points Completed in this iteration: 14

Iteration 2:

Accomplishments:

1. Meeting with client and demo iteration1 changes (Login and signup features)
2. Create models as per the DB model diagram at https://lucid.app/lucidchart/901b0e43-df07-48df-bdc9-

1ae8d9c21f7c/edit?invitationId=inv_9b5c53ff-65f9-4977-80c7-e0b6e4731c41&page=J960fNmy6zCK#
3. User stories completed:
    a. User Story: As a host, I want to create a event
    b. User Story: As a host, I want to add and customize questions in events
    c. User Story: As a host, I want to add points to the answer options.
    d. User story: As a host, I want to see a list of all events I have created

Points completed in this iteration: 16

Iteration 3:

Accomplishments:
1. Presented demo of iteration 2 implementation and collected feedback accordingly
2. User stories we implemented in this iteration are:

    1. User Story: As a host, before the event, I wish to be able to click start to allow participants to now start an event.
    2. User Story: As a host, before the event, I wish to see how many people have joined the event
    3. User Story: As a host, during the event, I wish to see the current questions users are answering.
    4. User Story: As a host, before the event, I wish to display a QR code that people can use to join
    5. User Story: As a participant, I wish to see the questions answered so far and how many are left, so that I can know how many questions do I still need to answer
    6. User story: As a participant, I want to be shown a QR code, that I can scan to join a live event
    7. User Story: As a host, I want to host the event live
    8. User story: As a participant, I want to see a question and possible answer choices, so that I can answer a question and participate in the study.
    9. User Story: As a host, before the event, I wish to give participants a link to join the event.
    10. User Story: As a host, during the event, I wish to be able to click next to show the next question.
    11. User story: As a participant, I want to be shown a link that I can use to join a live event.

Points completed in this iteration: 36

Iteration 4:

Accomplishments:

1. Presented demo of iteration 3 implementation and collected feedback accordingly
2. User stories we implemented in this iteration are:
    a. User Story: As a host, during the event I wish to see the overall position of the different users on the privilege walk
    b. User story: As a participant, when I finish the activity I wish to see my final position on the graph in relation to other participants
    c. User Story: As a participant, I wish to see how many people have responded to a question so far, to see how many people are participating in the study.
    d. User Story: As a host, during the event I wish to see the distribution of answers for the current question.
    e. User Story: As a host, once everyone has answered all questions, I wish to be able to see the position of everyone on the graph.
3. Performed end-to-end testing of the product and made cosmetic changes
4. Presented the final demo to the client. Client is happy with the product
5. CI/CD pipeline and AWS deployment
6. Created Final project report
7. Project demo video, poster and poster demo video

Points completed in this iteration: 38

Customer Meetings/Demos

Please find below the customer meeting dates and a short description of what was discussed.

February 24, 2022 4pm-5pm.
User study with Professor Dilma Da Silva and our client Dr. Thomas. They gave us feedback on our lo fi-prototypes.

March 8, 2022 4:45pm-5:15pm.
Discussed with our client the feedback from Professor Dilma Da Silva and what parts of the feedback we should implement. Also finalized discussion on the technology stack we would be using.

March 15, 2022 4:45pm-5:15pm
Informed client about our progress so far. We had nothing to demo at this point because we were mainly doing project setup, configuration and deployment process.

March 21, 2022 10:30am-11:00am

User study with Dr. Wilkinson, Heather H. During this study we learnt about the different possible departments at Texas A&M that might be interested in our solution, for example Professors in the Psychology department might be interested in our solution.

March 22, 2022 1:30pm-2:00pm
User study with Dr. Jonan Donaldson. During this study we mainly learnt that some people might take offense to using the word “Privilege Walk” and instead we should consider using a different word such as “Perspective Walk”

March 22, 2022 4:45pm-5:15pm
Met with our client and aggregated feedback from all our user studies and agreed on the set of features for the MVP. We agreed on what we would prioritize based on the feedback we got. For example, we agreed not to include the Walk graph while participants are answering questions.

April 5, 2022 1:30pm-2:00pm.
Demoed Iteration 2 user stories and got feedback from the client.

April 12, 2022 1:30pm-2:00pm
Demo iteration 3 user stories. Client also suggested we add a timer to the host perspective walk page as users are answering questions.

April 26, 2022 1:30pm-2:00pm
Demo Iteration 4 user stories. Demoed the final product and general feedback was the client was satisfied with the final product.

Testing Process (BDDs/TDDs)

Explain your BDD/TDD process, and any benefits/problems from it.

Frontend:

TDD:
Used Jest testing framework and React Testing Library which is utility for testing components.
The React libraries used for TDDs are ‘jest’ and ‘react-testing-library’.
TDD tests are located in ‘src/components/__test__/’ directory as ‘.test.js’ files.
Snapshots of the UI components can be found in ‘src/components/__test__/__snapshots__/’ directory.
Command to run all unit tests: `npm test`

Command for test coverage: `npm run test:coverage`
Benefits:

- Code has fewer bugs and errors
- Produces higher code coverage and therefore better code quality

Problems:

- Had difficulty in mocking web sockets for unit testing

BDD:
Used Cypress which is a front-end test automation framework. Integrated that with Cucumber which is a testing tool that supports Behavioral Driven Development (BDD). The React libraries used for BDDs are ‘cypress’ and ‘cypress-cucumber-preprocessor’. BDD tests are organized under ‘cypress/integration/’ directory as ‘.feature’ and ‘.js’ files. Feature files contain the descriptions of BDD test scenarios written in Gherkin language, while the step definitions are written as javascript files.
Command to run BDD tests: `npm run test-bdd`
Report generation:

- Run the command *node cypress/cucumber-json/cucumber-html-report.js*
- Reports will be available at ‘cypress/reports/’ directory
- Open ‘cypress/reports/index.html’ to view the report

Benefits:

- Can easily share tests with teammates and clients to show how app’s behavior under specific conditions
- Features can be easily translated as BDD tests
- Automation of end-to-end testing of various scenarios

Problems:

- Had difficulty in mocking web sockets for integration testing

Back-end:

TDDs:

For performing the TDD tests on the Django-based back-end, we have used the testing framework out-of-the-box, which is based on Python's unittest library. The tests for all the individual app components within Django are included in the ‘tests/’ directory within each of the apps. The framework allowed us to simulate a client and make calls to the REST APIs as well as the WebSockets.

Here's how you can run the tests:

- To run the tests without generating coverage reports (recommended for CI), just run `python3 manage.py test`.

    - In order to test one specific app within Django, you can add the app's name in the above command like `python3 manage.py test <name_of_the_app>`.
    - For instance, if you want to test only the app named user_mgmt, you must run the following command: `python3 manage.py test user_mgmt`.
- To run the tests with **coverage reports** (recommended that you do this on your local environment), run the following commands:
    - This command runs the tests and calculates the coverage: `coverage run --source='.' manage.py test`.
        - You may also test a specific app by just adding the app's name in the above command like `coverage run --source='.' manage.py test <name_of_the_app>`
    - To view the coverage report, please run the following command after running the command above `coverage report`.

Benefits:

Making the TDDs and having good coverage on them was very beneficial in two ways:

1. Making modifications to existing features:

   When we were modifying the APIs that we had programmed in the previous iterations (for example, adding the graph settings with the event details API response), we often had to change the functions and the serializers that were used by multiple APIs. This would lead to bugs or cause the other APIs that used the same serializers to crash. The tests helped us identify any breaking changes.

2. Halting the deployment of a bad version of the app

   The CI/CD pipelines that we have set up on GitHub for the back-end will run the TDDs first and checks if all of them have passed, before deploying the new version of the app to the AWS server and replace the currently running version of the application.

Problems:

We did not face any problems by having TDD test cases. TDDs have been very beneficial with the Django application. The only problem we faced was getting good coverage for WebSocket-based code.

BDDs:

The process:

BDD in this project is implemented using behave-django. Here's how it is implemented in this project:

- All the BDD tests are within the behave_tests directory, which is in the root folder of this project.
- The `.feature` files are within the features directory inside the behave_tests directory.
- The steps for processing the features are within the steps directory inside the behave_tests directory.
- The BDD tests can be run by using the command: `python3 manage.py behave`

Benefits and Problems:

The Django back-end only consisted of APIs and WebSockets. The web-based User Interface is all on the React application. Therefore, in order to simulate the user's behavior, most of the BDDs were written within the React application. The BDDs on the back-end were mostly written to test the expected results of the API calls, which would be beneficial if other applications would want to integrate Privilege Walk into them by using our APIs within their application (like how applications today use Google Maps APIs for getting a map into them or Stripe APIs to process payments).

Configuration Management

Front-end:

1. config/webpack.config.* - files contain the webpack configurations related to build folder, integration of plugins, loaders, etc.
2. .env.development - file contains environment variables used in the app
3. Babel.config.js - file contains configuration for babel presets and plugin integrations
4. Cypress.json - file contains configuration for cypress app urls, environment variables, and test files paths
5. Jest.config.js - file contains configuration for jest testing framework integration
6. .eslintrc.json - file contains configuration for eslint
7. .prettierrc - file contains configuration for beautifying the code and ensures the code format is same across all the code files
8. Package.json - file also contains configurations and scripts for various libraries and commands

Back-end:

1. There are three main documents on the back-end repository:
    a. The first one is the README.md file within the root of the project, which is a markdown file that describes how to install the application once cloned/forked and different ways to run the application (with and without

Docker). The README.md file also contains a brief description of how the application can be modified by the person forking/cloning it, and also how the BDD and TDD tests can be run.

   b. The two other documents are the API documentation and the WebSocket documentation which are written as markdown files within the 'docs/' directory:

2. Branch management:
   a. There were new branches created for any new feature that was being developed or a bug that was being fixed.
   b. The developer of that branch would need to raise a pull request if he/she wanted to merge those features into the main branch.
   c. Whatever would be committed to the main branch would automatically be deployed to AWS.
   d. Whenever a pull request was raised to merge a branch into the main branch, we set up the Github actions to run the TDD tests automatically:
      i. The results of these tests, along with other merge conflicts would be displayed on the pull request page of Github.
      ii. GitHub would prevent the merging if the tests fail or if there are merge conflicts.
      iii. Merge conflicts or the parts of the code that would be making the tests fail could be resolved and committed to the branch raising the pull request and then the pull request would run the checks again.
   e. When all the checks are passed, the pull request can be approved.
   f. When the pull request is approved, the feature or the bug fixture branch would be merged into the main branch.
      i. The merged branches were always deleted to keep the current working branches of the repository clean.
      ii. Therefore, currently, there is only one branch in the repository (at the time of writing this document)
   g. There have been 35 pull requests till now (one for each branch on average).
3. There have also been 5 releases (one for each iteration, plus one more for improvements after the 4th iteration) at the time of writing this document.
4. We also had one spike to test the basic WebSocket set-up on the back-end. For that, we implemented a simple chat application consumer on the back-end to verify the WebSocket setup. This was done because the basic WebSocket setups are complicated and it will be hard to verify the basic setup directly with the complex event control feature that was a part of the actual application.

Deployment and tools.

Discuss any issues you had in the production release process to Heroku.
Describe any issues you had using AWS Cloud9 and GitHub and other tools.
Describe the other tools/GEMs you used, such as CodeClimate, or SimpleCov, and their benefits.

Some of the tools we used include:

Tool	**Benefit**
Code Climate	Helped us analyze the quality of our code for example, are we following the DRY principle?
Cypress	Helped us with BDD testing on the frontend.
Jest	Helped with frontend unit testing.
Django Unit test framework	Helped with unit testing our backend.
behave-Django	Helped us with unit testing our backend.
GitHub	Helped us with version control and having multiple developers work together on the same repository.
GitHub Actions	Programmed the CI/CD pipelines to automatically test and deploy the pushed code to AWS so that the developers don’t have to deploy them manually
Docker-compose	Helped run the back-end app within AWS, along with other dependencies such as PostgreSQL and Redis.
Amazon Web Services (AWS) EC2	Used for hosting and running the back-end, as well as the database and the key-value store required for Django channels.
Heroku	Used for hosting and running the front-end. We also used it to host the back-end in the beginning before WebSockets were implemented.

<u>**Issues with production release to Heroku**</u>

1. Heroku does not allow deploying multiple applications as Docker containers, but the back-end also needed a Redis key-value store for the Channels layer. So, we had to move the back-end to AWS

2. Recently, Heroku has had a platform bug with automatic deploys from Github. So, we had to deploy the front-end manually to Heroku

Repository Contents and the Deployment process

Front end**:**

1. Src - folder contains the component-wise code, their unit testing files, Redux store files, and utility functions.
2. Config - folder contains the webpack configuration files for ‘dev’ and ‘prod’ environments.
3. Cypress - folder contains the BDD feature files and step definition files.
4. Package.json - contains the dependencies for the frontend app.

Deployment process:

1. Dockerfile - Github hooks trigger the deployment pipelines which build a docker image from this file.
2. Heroku.yml - file containing commands to build the docker image from the docker file and run it.
3. Automatic deployment for the main branch on GitHub is setup on heroku. Therefore, everything that is committed to the main branch will be deployed automatically to Heroku.

Note: Recently, Heroku has pushed an update to their platform. They have an internal platform bug that is preventing automatic deployments from Github. So, the React app will have to be deployed manually to Github using Heroku CLI

Back end:

1. **.github/workflows -** Contains all the CI/CD pipeline instructions for different scenarios such as pull requests and commits to the main branch, including the commands for running the TDDs automatically, as well as deploying the application to AWS.
2. **behave_tests -** Contains all the BDDs under ‘features/’ and the program for executing those steps in the features under ‘steps/’
3. **docs -** Contains the markdown files for API documentation and the WebSocket documentation.
4. **host -** It is a Django application that handles all the APIs related to the event host side of the application
    a. **host/controllers -** contains the code for all the APIs.

    b. **host/migrations -** contains the Django-generated scripts to transfer the models ans the modification to the models to the database.
    c. **host/serializers -** contains the code for the serializers that can be used to assist the APIs to translate JSON-like dictionaries into model queries for storing and loading information.
    d. **host/tests -** contains the code for all the TDD-based test cases.
    e. **host/utils -** contains the functions that can be used by multiple APIs to perform the same kind of task.
    f. **host/admin.py -** code for linking the models to the Django admin dashboard.
    g. **host/apps.py -** Django generated (for letting the settings know that this directory contains an app).
    h. **host/models.py -** The database models for the host side of the app.
    i. **host/urls.py -** Contains the code for linking the URLs to the host side APIs.
5. **privilege_walk_be -** The main application of Django that contains the settings and other basic files required for running the Django application.
    a. **privilege_walk_be/asgi.py -** Contains the Asynchronous Server Gateway Interface code for running the application with websockets enabled and linking them to a Daphne execution server.
    b. **privilege_walk_be/settings.py -** Contains the settings and basic configurations for the Django application to run such as the links to the database and the key-value stores in different execution environments, as well as the dependent middlewares and applications.
    c. **privilege_walk_be/urls.py -** Contains the base (root) URLs for http(s) and ws(s) protocols, as well as the code to link them to different applications' sub-URLs.
    d. **privilege_walk_be/wsgi.py -** (not used anymore after implementing websockets) Contains the Web Server Gateway Interface code for running the application without websockets enabled and linking them to a Gunicorn execution server.
6. **user_mgmt -** The django application for handling the authentication of hosts as well as handling the anonymous users who participate in the events.
    a. **host/controllers -** contains the code for all the APIs.
    b. **host/migrations -** contains the Django-generated scripts to transfer the models ans the modification to the models to the database.
    c. **host/tests -** contains the code for all the TDD-based test cases.
    d. **host/admin.py -** code for linking the models to the Django admin dashboard.
    e. **host/apps.py -** Django generated (for letting the settings know that this directory contains an app).

    f. **host/models.py -** The database models for the host side of the app.
    g. **host/urls.py -** Contains the code for linking the URLs to the authentication APIs.

7. **walk -** The django application that contains the code for handling the live privilege walk events.
    a. **host/controllers -** contains the code for all the APIs and the WebSocket consumer to be used during the live event.
    b. **host/migrations -** contains the Django-generated scripts to transfer the models ans the modification to the models to the database.
    c. **host/tests -** contains the code for all the TDD-based test cases.
    d. **host/admin.py -** code for linking the models to the Django admin dashboard.
    e. **host/apps.py -** Django generated (for letting the settings know that this directory contains an app).
    f. **host/models.py -** The database models for the host side of the app.
    g. **host/urls.py -** Contains the code for linking the URLs to the host side APIs and the WebSocket consumer.
8. **.behaverc -** contains the configuration for the BDD command to execute.
9. **.coveragerc -** contains the configuration for the TDD coverage to follow.
10. **.coveragerc -** contains the files to be ignored by the Git VCS.
11. **Dockerfile -** contains the instructions for Docker to set-up and execute the basic back-end application within a container.
12. **LICENSE -** The MIT license that defines the terms and conditions of usage and modifications to the open-sourced application.
13. **Procfile -** (Not used anymore) contains the execution command for Heroku deployment.
14. **README.md -** The markdown documentation file that defines the steps required to run the application (with and without Docker), including installing the dependencies.
15. **docker-compose.yml -** Contains the instructions to start the docker container of the application, along with the containers of other dependencies such as Redis and PostgreSQL.
16. **manage.py -** Generated by Django to facilitate the execution of various scripts and commands related to database migrations, user management, running the test server, and also running BDDs and TDDs.
17. **requirements.txt -** contains all the dependencies (python packages) required to be installed in order for the application to run
18. **run.sh -** contains the commands required to run the application (with the data migration first) in production. This is mostly used within the docker container.

19. **run_heroku.sh -** (not used anymore) contains the Gunicorn command to run the application on Heroku.

CI/CD:

- Commits and merges to the main branch:
    - First, the TDDs are run on the committed code.
    - If the code passes the TDDs, a docker image is built and pushed to Docker's free container registry.
    - Then GitHub actions connects to AWS and deploys the latest docker image pushed to the registry
- Pull requests made to the main branch:
    - The TDDs are run in order to let the developers know if the feature that they have pushed is causing another feature that was already deployed to crash.

Links

Pivotal Tracker: https://www.pivotaltracker.com/n/projects/2556087
public GitHub repo:
Backend: https://github.com/Privilege-walk/back-end
Frontend: https://github.com/Privilege-walk/front-end
Documentation: https://github.com/Privilege-walk/privilege-walk
Backend is running on AWS: http://54.157.248.16:8000/
Code Climate Frontend: https://codeclimate.com/github/Privilege-walk/front-end
Code Climate Backend: https://codeclimate.com/github/Privilege-walk/back-end
Heroku deployment: http://privilegewalk.herokuapp.com/ (this is updated link)

Video Links.
NOTE: In the poster video we refer to the privilege walk as Empathy Walk activity. In reference to these videos "Empathy Walk" and "Privilege Walk" can be used interchangeably and refer to the same activity.

- Poster video **:** https://youtu.be/kQtJN9rcCzg
- Demo video: https://youtu.be/cQ3h20ebQ70
- Combined video of poster and demo: https://youtu.be/wslAbMgIf7k