ENGO 551: Final Project Report

Izhan Alam

Project Report: ENGO 551

Introduction:

The web application created for this project is titled 'Event Maps'. The purpose of this web application is to allow users to see which events are currently happening near their area. Users are able to create an account and sign in. Once signed in, they are able to create an event that is currently happening or will happen in the future. If an event is in the future, then the hyperlink to access the comments and additional details of the event are disabled until it goes live. All of these events have a timer and will disappear from the map once the event expires. In addition, each event on the map has a pop up which you can click to comment on the event, view the details, or get directions to the event. In addition, there is also a private instance of the map which is unique to a user. A user is able to create events on this private map which last for 24 hours. They are able to share a link with other users which allows them to see events they have created which is helpful in planning something like a trip. Others are also able to comment on the private map instance to share their thoughts. A user is also able to select the icon for which the popup uses to make sure it matches their event functionality.

Architecture:

Web architecture generally refers to the interactions between an application, a database, and middleware systems on the web to ensure that multiple applications are able to work together in sync. This can be broadly categorized into two different classes consisting of the UI/UX application components, and the structural components. The UI/UX component refers to the interface layout of a web app, whereas the structural component refers to the client and server sides. The client side of the structural component exists on the user's web browser and does not need any adjustments. On the other hand, the server side consists of the app logic and the database. Architecture can be further divided into single-page applications, microservices, and serverless architectures.

The website built for the final project falls under the single-page application, in which, it does not load a completely new page from the server unless needed. This directly impacts the user experience in which the user does not need to wait for any server load time until submitting their data. The dynamic interaction is done primarily through JavaScript, which leads into the client-side architecture.

Client-Side Architecture:

The client-side architecture for the website was written with three different programming languages consisting of HTML, CSS, and JavaScript. This is also referred to as the front-end side of a web application. In this case, the code is loaded into the browser and responds to the user input. It can also be seen and modified by the user. However, it does not have the ability to read a server directly.

The HTML portion of my GIS website consists of eight different HTML pages, and one HTML template, which the other pages are based off of. The HTML pages are comprised of the about page which details the information about the website, the sign up page, the log in page which details whether or not the login was successful, the search page which shows the results of an input search, the index page which is the default homepage, the account created page which details whether the account was created successfully, the private map page, and the events page. It should also be noted that jQuery is implemented in these HTML pages to allow for a more interactive user experience depending on the

page. The main uses of these were to display the comments/directions. These are hidden until a user clicks on a button to allow them to be displayed.

Of these pages, the private map page and the event pages are dynamic. A user can opt to create their own private map and share this with their friends, and thus the page is created dynamically based on the user's ID. The event page is created for every event which is created on the map which showcases information about the event such as the title, description, location, user comments, time until event closes, and directions. The URL path for each of these pages can be found in the code, under default/urls.py.

There are two main JavaScript files which is the JavaScript ran on the public map (and public pages), as well as the JavaScript which is ran specifically for the private maps. The main differences in these the way they access the server to load up the pop ups and event information on the maps which will be explained later on in the server-side architecture.

All the HTML pages require some sort of authentication to view/access different features of the web application. The template HTML does most of this authentication. For example, on the homepage, you will be greeted with a sign-in form if you are not logged in, and a 'Create an Event' form if you are signed in. This leads into the next portion, server-side architecture.

In addition, there is a portion of the JavaScript is used to verify dates (local, time zones, etc.), and verify whether an event is currently active or will be active in the future for the user based on their current location. This is a bit different from the application logic which verifies the timer and determines when the event expires.

Server-side Architecture:

Server-side architecture refers to the back end of a web application. In this case, the code is on the server and it responds to the HTTP requests. It is responsible for creating the user-requested page as well as storing relevant data. This means, there are two portions of this architecture consisting of the application logic (how the website behaves when a user enters information), and the database which stores the information. This portion of the report will deal mainly with the application logic and how it works. First and foremost, there are multiple frameworks and languages which you are able to choose from but for this project, I chose to use the Django framework in conjunction with python. The reason for this is primarily because it is a much heavier back-end framework and has much more features than other frameworks like FLASK.

Most of the server-side logic code can be found under default/views.py. This consists of many different functions. The first notable function is used to create the account. This checks the username and email with the database and creates a new account when the username/email is available. The passwords are always hashed, so they cannot be directly seen from the database. The log in function verifies your credentials and authenticates the user if the verification is correct. This function checks the hashed password. When the authentication is complete, the user is given a session ID. The session ID essentially saves the user to a cookie. This is important to allow the user to view the authentication required portion of the website without needing the user to continuously sign in every time. In addition, I made the sessions cached for better performance. Some portions of the website are also cached which allows

for a better client-side architecture, in which the user will not have to wait as long for the website to load, as portions of it would be pre-loaded by the cache.

The most notable function for my website was the 'createEvent' function. This function gets the latitude, longitude, time, and icon from the form, and creates mainly two things. This function gets form data (the latitude, longitude, time, and icon) and essentially saves it to the database. In addition, it creates a geojson feature with information about the event and appends this to a geojson feature-collection which is stored in the database. This is based on the timer selected, so each timer has a different feature collection. Once the event is created, the 'event' function takes care of the redirection to the dynamically created event page based on the event's ID. It also serves as a way to comment on the event when the user is authenticated.

Similarly, there are functions to create the private events, and private comments. However, the private events do not take you to a dynamic web link created just for the event as it is not needed. In addition, private events do not have comments associated with each event, but rather the private map itself which is based on the user's ID. The events created on private maps are saved to a geojson feature and appended to a private geojson feature collection. There is also a function which removes the events after 24 hours have passed.

The main app logic is built on geojson features. When an event is created, there are multiple functions in views.py which check for the validity of the event. Once a feature is validated, it is appended to a feature collection. This feature collection is saved to a URL link which the JavaScript reads as a json object and then displays the features on the MapBox map. Every time the JavaScript reads the link, there is a function which will compare the published date against the expiration date and determine whether its recent. This also updates the geojson link which will then update the map every time it is loaded.

Database Design

The database was created using Django models which can be found under default/models.py. There is a total of 18 tables (models) in the database. However, some of these tables are for administrator privileges, and permissions which are auto generated from Django. The web application utilized mainly two of these auto-generated tables which consisted of registering and saving a user's info, and the session of the user. On top of this, an additional 8 models were created for the web application itself to store various other information.

User Registration:

User registration is done through the table titled 'auth_user'. The table has 11 columns which consist of the ID, password, last login, super user status, username, first name, last name, email, staff status, activity, and the date the account was created.

	id [PK] integer	password character varying (128)	last_login timestamp with time zone	is_superuser boolean	username character varying (150)
1	1	pbkdf2_sha256$180000$j...	2020-04-17 18:12:29.007289+00	false	izhan
2	2	pbkdf2_sha256$180000$...	2020-04-10 12:12:12.437939+00	false	account2
3	3	pbkdf2_sha256$180000$l...	2020-04-11 02:45:17.920129+00	false	newaccount

first_name character var	last_name character v	email character varying (254)	is_staff boolean	is_active boolean	date_joined timestamp with time zone
		izhan	false	true	2020-04-10 11:15:41.626001+00
		account2	false	true	2020-04-10 11:37:52.851453+00
		newaccount	false	true	2020-04-11 02:16:13.283773+00

The two figures above showcase the table with some information filled. The first and last name of the person is left as null, as the web application did not require it during registration. The password is automatically hashed, and there is no way of knowing the user's password without de-hashing it. Users created on the web application cannot be super users or staff, hence the default status is false. If a user were to be a super user or staff, some permissions which are not normally available to users would be granted to them.

The second model/table concerns itself with a user's session. It effectively is used to track which users are currently logged in on the website and authenticates the user. The cookie which is associated with client-side contains a session ID for identification rather than the data itself. The session is saved on the backend.

	session_key [PK] character varying (40)	session_data text	expire_date timestamp with time zone
1	1dveatzysd001kqn9zwfoavyle58z...	MDBjZjU5NjUyYTI...	2020-04-24 12:12:12.510956+00
2	6un0ian9amjn284p1fnx1uq08hfgl...	NTc3ZjAwZGQ1M...	2020-05-01 18:12:29.090306+00
3	jo7sk2bla07zxx1331ra0x9thraznk...	YzI5ZTA0ZGY1NjE...	2020-04-25 02:45:17.997147+00
4	w0v8zkbvacswe68dfgwdh8pco69...	ZTM1ZmFkZDBjNj...	2020-05-01 18:07:06.391255+00

Now that the registration and authentication of the user was dealt with, the next task was to create the tables which store an event's information. This is done by creating a new table and linking the user as a foreign key so that the detail of the user who created it is known. The figure below showcases the columns associated with the table.

	id [PK] integer	latitude character varying (30)	longitude character varying (30)	pub_date timestamp with time zone	title character varying (100)	comments character varying (280)	time integer	user_id integer
94	94	51.030451591313124	-114.11206876220693	2020-04-26 03:00:00+00	9:00	Test	1800	7
95	95	51.06853881948143	-114.0002226084214	2020-04-26 01:00:00+00	Party	Test	1800	7
96	96	51.093025569885725	-114.10863553466766	2020-04-26 02:00:00+00	8:00	Event expires at 8:30	1800	7

When an event is created, the latitude, longitude, publication date, title, description, time and user id is saved. The publication date is dependent on whether the event is for the future or present. If there was no date selected for the event, then the current local time is used as the publication date. Otherwise, the time submitted in the form is used which is then converted to local time. The time column is for the timer of the event in seconds. For example, 1800 seconds is equivalent to 30 minutes. This is used to determine whether an event is currently active, will be active in the future, or if it is expired as explained in the architecture section. The latitude and longitude are obtained by moving a marker on the map and is used to create the position of the pop-up.

The next few models are used to verify whether the event is active and also used to store the geojson objects which are used for the geojson link in which the map gets its data from. There are five of these tables, but they all follow the same general format. The only exception is that the function which verifies whether an event is currently active or expired changes based on the timer of the event.

The general structure is shown in the figure below.

	id [PK] integer	geojson_object text	feature text	pub_date timestamp with time zone	recent boolean
1	81	{"type": "FeatureColl...	{"type": "Fe...	2020-04-25 12:48:38.740353+00	false
2	82	{"type": "FeatureColl...	[null]	2020-04-25 12:48:38.883385+00	false
3	83	{"type": "FeatureColl...	{"type": "Fe...	2020-04-25 12:54:28.29704+00	false

This table contains a feature collection and a feature. Each time an event is created, a feature is also created in the following format:

{"type": "Feature", "geometry": {"type": "Point", "coordinates": [-113.959633, 51.011015]}, "properties": {"description": "future event:9:00", "URL": "http://127.0.0.1:8000/event/103", "comment": "ww", "icon": "hospital-15", "date": "2020-04-25 21:00:00-06:00", "current_date": "2020-04-25 19:51:05.269798-06:00"}}

This is for a single event. Note that here, there are two dates. The variable ‘date’ is when the event goes live, and the variable ‘current_date’ is when the event was first created. The JavaScript uses this as a check to confirm whether an event is for the future or the present. The recent column is based on the following function contained in each of the model:

```
def was_published_recently(self):
        return self.pub_date >= timezone.make_aware(datetime.datetime.now(),timez
one.get_default_timezone()) - datetime.timedelta(hours=6)
```

This is simply a Boolean which returns true or false based on whether the time the event expires is greater than the time the event goes live. Once the event expires, the recent column is updated to false.

This update is then sent to the URL containing the geojson from which the map loads and is removed when the recent column is updated to false.

In addition, each feature is appended to a 'FeatureCollection'. The following is an example:

{"type": "FeatureCollection", "features": [{"type": "Feature", "geometry": {"type": "Point", "coordinates": [-114.094559, 51.090007]}, "properties": {"description": "Party", "URL": "http://127.0.0.1:8000/event/70", "comment": "Test", "icon": "attraction-15", "date": "2020-04-25 12:00:00-06:00"}}, {"type": "Feature", "geometry": {"type": "Point", "coordinates": [-114.094559, 51.090007]}, "properties": {"description": "Party", "URL":……

This feature collection is what is sent to the geojson URL which the map then reads and creates the markers for in the JavaScript as outlined in the architecture section.

The table for creating a private geojson feature for the map is as shown in the figure below:

	id [PK] integer	geojson_object text	feature text	pub_date timestamp with time zone	recent boolean	user_id integer
32	32	{"type": "FeatureColl...	[null]	2020-04-17 13:07:08.788095+00	true	6
33	33	{"type": "FeatureColl...	{"type": "Fe...	2020-04-17 13:10:09.186379+00	true	6
34	34	{"type": "FeatureColl...	[null]	2020-04-17 13:10:09.332418+00	true	6

The main difference is that each FeatureCollection (the geojson URL) is tied to the user ID. This means that since each geojson URL for a private map is dynamically created, each URL contains a different FeatureCollection based on what URL the event was posted on. The user_id is simply the instance of the private map for which the feature collection exists for.

The next table is used for comments for each event. The structure is as follows:

	id [PK] integer	author character varying (200)	event_comments text	likes integer	dislikes integer	time integer	popup_id integer	user_id integer
1	1	testaccount	Got my car, Thanks!	0	0	0	33	4
2	2	izhan	Test Comment	0	0	0	44	1
3	3	account5	I'm planning to the stu...	0	0	0	50	7

The basic idea is that there are two foreign keys which relate the event ID (labelled as popup_id) and the user (labelled as user_id). A user is allowed to comment multiple times on an event.

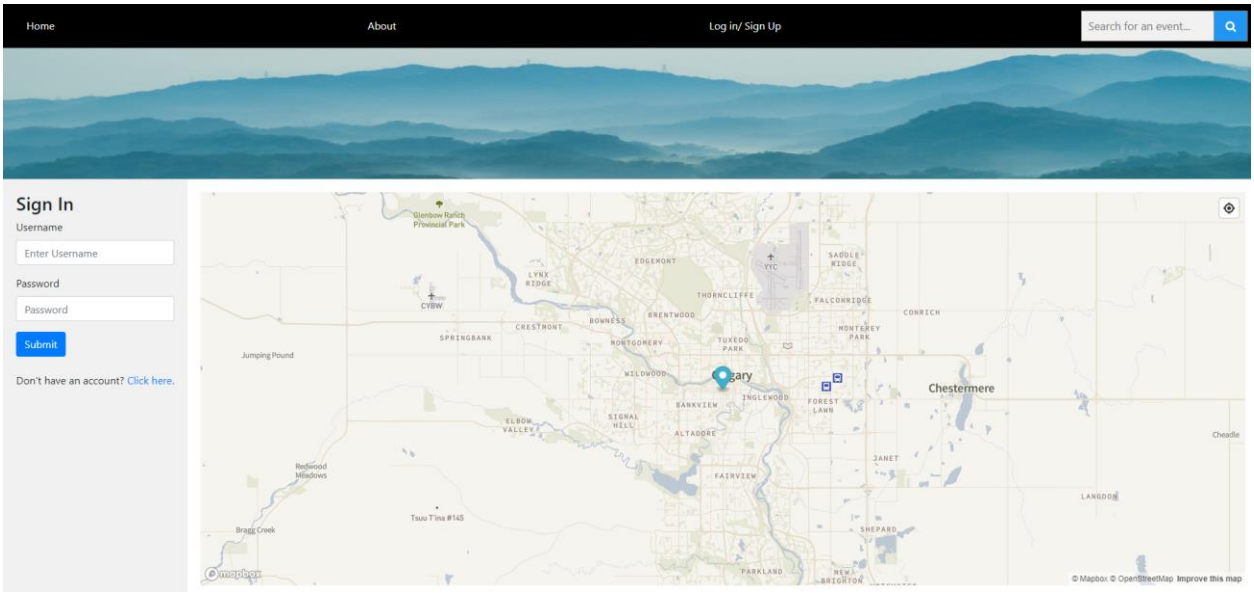
Similarly comments for private maps are stored in another table as follows:

	id [PK] integer	comment text	pub_date timestamp with time zone	instance integer	author character varying (280)
1	1	xx	2020-04-10 10:58:23.037742+00	2	Anononymous
2	2	xx	2020-04-10 11:02:32.824614+00	2	Anononymous
3	3	zz	2020-04-10 11:03:04.908792+00	2	Anononymous

In this case, the instance is tied directly to the private map which as explained in the architecture is linked to the user's ID. Users do not need to comment using their own username and can comment as anonymous if they choose to do so.

Design Choices:

The web application was designed with simplicity in mind. It follows a blue and white colour scheme throughout the entirety of the website. The following figure shows the basic layout.



UI/UX design:

The main focus was on creating a simple user interface. This is done usually by keeping the interface simple and creating consistency. Consistency was achieved by making the input buttons and text fields simple and easy to read. Labels are used for each input, so the user knows what each input field does. The map takes the majority of the website at it is the main focus of the web application. There is a side column at the left of the map, which is used to create an event, sign in, or create a comment. This means that throughout the different webpages, this main layout will remain consistent so the user will always know that inputs are to the left, and the map will always be to the right. There is no clutter in the webpage to ensure that it remains simple and clean. Each item in the webpage has a purpose. To ensure easy navigability throughout the different webpages, there is a navigation bar at the top of the web

page. This allows the user to click on the page they wish to go. The search bar allows users to easily search for an event based on the description or title of the event. On top of this, when the search is performed, you are also able to access expired/archived events if you wish to do so. The search results have hyperlinks which allows a user to easily access more details of the event if they wish to do so.

In addition, when you sign in, the layout automatically changes to ensure unnecessary things such as the sign in or sign up page are removed which further improves user interaction. The web application also makes extensive use of jQuery to hide unnecessary information unless the user has explicitly requested it. The links are always a hyper link to make sure that there is no need to copy and paste the URL into the address bar. To make sure the user always knows what is happening, a message will appear in the banner. For example, after signing in, the banner will have a message saying, 'Logged In'.

The goal of the UX design was to allow it to be accessible to all demographics and easy to understand. This was effectively done through the UI design as mentioned above. In addition, the web application is made so that cookies and sessions are efficiently used throughout the website so the user will not have to worry about resubmitting a form or having to sign in more than once to access a page. Even if the web browser is closed, the session remains intact until it expires or until the user logs out.

API Design:

The main API used for this web application was based on MapBox's API. MapBox was used over Leaflet as I believe MapBox has much more customization for the map than Leaflet does. In addition. MapBox had some additional API's which I used such as the directions API.

First and foremost, MapBox studio was used to make the map look better than the default map. MapBox's API was then implemented in the web application to display the map. The driving directions for each event were also based on MapBox's directions API. A geolocation API from MapBox was used if a user wanted to use their current location instead of dragging the marker around the map. The other notable API used was jQuery, which was used to stylize some of the web pages and get the current time.

In addition to utilizing the previous APIs, I created an API for the website itself. It is a geojson response function API which showcases the current active events on the map. Anyone is able to use this API to get the current active events. This is also the API response which the web application uses itself to display the features on the map. For instance, the link (hosted on local server) for events with a 30-minute timer is given as:

http://127.0.0.1:8000/geojsondumps30min.geojson

When the link is clicked, it will return a geojson response as such which shows current active events:

{"type": "FeatureCollection", "features": []}

Questions:

From the lectures, we learned that web 1.0 was a static internet. This meant that most of the webpages were read only and lacked user interaction. For the web application I created, this part only exists in the about page. The about page is the only read-only portion of the website and is what web 1.0 is. Web 2.0 expands on this and allows users to create and write content on the website. The web application lines up with web 2.0 very heavily. Almost all of the content which is created on the website is based on a user creating an event, and having other users participate in that event. This can be done through the form of attending the event in real life, or by adding comments to the event. It is heavily reliant on user submissions. Without user participation, the website would not be able to run. In addition, web 2.0 focuses on a user interacting with the website. This is also relevant in the web application, where the user is able to interact with the map and see the contributions which they have made to the map.

As web 2.0 grows, the next step is to capture the data. The underlying goal of this website was to utilize the data from users for optimization. For example, a user would be able to create an event titled pothole, and people would be able to view this. From this simple event, you would be able to find out how many users are impacted by this pothole and thus whether or not it should be a priority for the city to quickly fix. The geolocation API which was implemented also has the underlying use of tracking where a user is. By tracking a user's location, you would be able to find out the amount of people that are currently at an event or at a particular location. Similarly, you would be able to find out the number of people which are visiting an event at a particular location and be able to see which locations are ideal for things like a store. The idea of this is called the information shadow. The more you are able to capture the data from the collective mind, the more information you are able to gather and utilize. MapBox uses OpenStreetMap which means that different data subsystems are cooperating, and by including these into my own web application, I am in turn advertising for them (if the website were public).

In the first lecture, we learned about the Internet of Things. The main point is that everything can be connected to the internet and interact with each other such as a smart fridge. What web 2.0 informs us is that the internet of things can work based on a large amount of sensor data. Not everything needs to be connected to the internet. Once machine-learning applications are able to understand sensor data, then any object can be easily connected to the internet. For example, in this web application, another feature would be that people would be able to upload pictures of the event they are at. If a machine-learning algorithm would be able to identify the number of people at the event based on the photo alone. If a user submitted an event about a traffic accident, the machine-learning would be able to immediately notify authorities. It is the idea that not everything necessarily needs a unique IP and be connected to the internet, and data from sensors such as your phone can be processed by machine-learning which would effectively connect the item to the internet.