High level design:

Event collector has been divided into 3 subcomponents:

* Geolocation
* Events
* Errors

Blueprints of these components have been created and registered with the app to allow scalability

**Database used for this project is sqlite. SQLAlchemy is used to interact with the database**

**Geolocation:**

This component handles and stores all requests - given an ip, identifying the location and storing into the database

**Routes.py:**

This module handles the API s for :

/get\_geoloc – This api allows user to enter an ip address. This checks our database if the data exists for the ip, if it does, it calls next api. If the data does not exist, it fetches the data geoip2.database library. The database *GeoLite2-City.mmdb is shipped as part of Event Collector*

/get\_geoloc/<string:ip> - This api renders the page containing the details of the location for the ip entered previously

*/search – This api lets users to enter data about location , from time and to time to query events based on these values. Once the user enters and submits, next api is called to display the data*

*/search\_result/<string:location>/<t1>/<t2> - This api queries the database for details as per the query and renders the html with the data*

**Forms.py:**

This module handles form fields required by various forms required for the user to enter data. Like ip address in the get\_geoloc api and location,from and to time in the search api

**Utils.py:**

This module contains the utility function to actually do the computation of getting the geo location from ip if the data is not already present in our database.

If an invalid ip is provided it throws a **HTTP 400**

**Events:**

This component handles the event collection functionality

**Routes.py**

This module handles the APIs for:

*/collect\_event/<string:ip> - This api accepts the event name from the user and calls the* ***collect\_event\_source*** *method that collects the events from the selected source. Source ip is passed as a parameter to the api. This api gets the job id and returns it to the user with the status ‘SUBMITTED’ .*

*/job\_submitted/<jobid> - This api gives the user the status of the job submitted. User can run this multiple times to know the status.*

**Forms.py**

This implements the class required for the form to enter event name by the user

**Utils.py**

This module contains the actual code to collect events from the source ip and update the database with the data. This first inserts a row into Jobstatus table and returns the id to the user which user can use with the /job\_submitted api to query it’s status

Later this runs the actual event collection in the background and updates the events table with the event data

This also limits the number of jobs to 5. When a 6th job comes in it throws a HTTP 429

**Errors:**

This component handles the exceptions that might occur .

Two errors it handles are:

429 – where it limits the number of submissions at a time to 5.

400 – where it handles invalid request

**Config.py:**

This is outside of the components described above. This module contains configuration details which is passed during app creation.

This contains secret key as it is required for WTF and also connection details to database. We can plug in a different database if needed here.

**Models.py**

This class inherits db.Model class of SQLAlchemy . This module implements the classes that corresponds to our database tables. This helps us in creating the tables as well by just importing the classes

There are three tables required for the project:

1. **Geolocation**(id, ipaddr, country, geolocjson, events)

events is a relationship between tables Geolocation and Events

1. **Events**(id, datecollected, time\_stamp, host,source, name, fields, ip\_address)

**Description:**

**Id** – unique id

**Datecollected** – date on which the event was collected

**Time\_stamp** – time at which the event occurred

**Host** – name of the host where it occurred

**Source** – name of the file/application that is providing us the data. E.g., in case of an event like ssh attack, this field will have the value /var/log/auth.log

**Name** – name of the event

**Fields** – actual event details. This can either be a simple string or JSON depending on the event

In this table, ip\_address is a foreign key to Geolocation table

1. **Jobstatus**(id, status, date)

This table tracks the jobs submitted and their corresponding status

**Templates:**

This folder contains all the html files needed for the project if it is run from a browser.

CSS under **static** folder and layout.html have been imported from another project of mine 😊

**Testing:**

Since the number of apis/links are very less we can use front end tools like selenium to automate testing from web browser

**Deployment proposition** has been mentioned in ReadMe.txt

**Potential Problems :**

Main problem with this implementation is handling security concerns. There needs to be some sort of mechanism to authenticate the user running the api. We can either use token based approach or username and password.

The way user provides credentials to the host from where data has to be collected also has to be encrypted before it is passed to the server.