



# EVENTRADAR

Robert Velaya  
Hyun Woo Kim  
Jonathan Manzano



# TABLE OF CONTENTS

1

## PROBLEM STATEMENT

You can describe the  
topic of the section here

2

## FUNCTIONALITIES

You can describe the  
topic of the section here

3

## TECHNOLOGIES

You can describe the  
topic of the section here

4

## DEMONSTRATION

You can describe the  
topic of the section here

5

## CODE

You can describe the  
topic of the section here

6

## CONCLUSION

You can describe the  
topic of the section here



# PROBLEM STATEMENT

Sure, here's how you can present the problem statement as bullet points for a PowerPoint presentation:

- **New to the City?** Struggling to find local events and community activities?
- **Navigating the Local Scene:** Discovering events in a new area is like navigating a maze—time-consuming and often incomplete.
- **Scattered Information:** Event details are spread across multiple platforms, making it hard to get a full picture.
- **Stay Connected:** Experience the heart and soul of the community with just a tap, ensuring you never miss out on what's truly special.
- **Our Solution:** A single platform that consolidates all local events, from farmers' markets to art openings.



# FUNCTIONALITIES

- User Accounts and Secure Authentication
- Event Aggregation Through API call
- Modular/Dynamic/Rendering Design



# TECHNOLOGIES

05

PYTHON

Programming Language

06

FLASK

Backend Framework

07

SQLITE

Database

08

JINJA2

Templating Engine  
Dynamically Renders HTML

09

BOOTSTRAP

CSS Framework

10

TICKETMASTER API

API

The background is a solid green color. It features several white dashed lines forming abstract, organic shapes. In the top left corner, there is a white arrow pointing left. In the top right corner, there is a white arrow pointing right. The text "LIVE DEMO" is centered in a large, black, hand-drawn font.

# LIVE DEMO





cs122\_final\_project - app\\_\_init\_\_.py

```
1 from flask import Flask, render_template
2 from flask_sqlalchemy import SQLAlchemy
3 from flask_login import LoginManager
4 # from flask_migrate import Migrate
5 from app.config import Config
6 from os import path
7
8 # Create the database instance globally
9 db = SQLAlchemy()
10 DB_NAME = 'eventradar.db'
11 # Initialize login manager
12 login_manager = LoginManager()
13 # Login manager needs a view to redirect to when a login is required.
14 login_manager.login_view = 'auth.login'
15 # Initialize database migration tool
16 # migrate = Migrate()
```

FLASK



# MODEL

```
PC  :: cs122_final_project - models.py

1  from datetime import datetime
2  from flask_sqlalchemy import SQLAlchemy
3  from flask_login import UserMixin
4  from werkzeug.security import generate_password_hash, check_password_hash
5  from app import db
6  #, login_manager
7
8  # User Loader for Flask-Login
9  # @login_manager.user_loader
10 # def load_user(user_id):
11 #     return User.query.get(int(user_id))
12
13  👤 Robert Velaya
14  class User(db.Model, UserMixin):
15      id = db.Column(db.Integer, primary_key=True)
16      email = db.Column(db.String(120), unique=True, nullable=False)
17      # username = db.Column(db.String(64), index=True, unique=True)
18      first_name = db.Column(db.String(20), index=True, nullable=False)
19      password = db.Column(db.String(100), index=True, nullable=False)
20      #relationship between user and event to store all events users sign up for.
21      # event = db.relationship('Event')
22
23  👤 Robert Velaya
24  class Event(db.Model):
25      id = db.Column(db.Integer, primary_key=True)
26      title = db.Column(db.String(128), nullable=False)
27      description = db.Column(db.Text, nullable=True)
28      location = db.Column(db.String(128), nullable=False)
29      # 4/18/2024 @ 1:37 pm check if utcnow is causing any issues
30      start_time = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
31      end_time = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
32      created_at = db.Column(db.DateTime(timezone=True), index=True, default=datetime.utcnow)
33      # user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
```

## Database

- + DDL
- ▼ eventrادر 1
  - ▼ main
    - ▼ tables 3
      - > event
      - > sqlite\_master
      - ▼ user
        - > columns 4
        - > keys 2
        - > indexes 3
  - > Server Objects



PC

cs122\_final\_project - auth\views.py

```
20 routes = Blueprint( name: "routes", __name__ )
21
22
23 @routes.route("/logout")
24 @login_required
25 def logout():
26     logout_user()
27     return redirect(url_for("views.home"))
28
29
30 @routes.route( rule: "/login", methods=["GET", "POST"])
31 def login():
32     if request.method == "POST":
33         email = request.form.get("email")
34         password = request.form.get("password")
35
36         # check if user used the field prompts properly -- cannot be empty.
37         if len(email) < 1:
38             flash( message: "Email cannot be empty, please try again", category="error")
39         if len(password) < 1:
40             flash( message: "Password cannot be empty, try again.", category="error")
41         else:
42             # checks if user exist through email. Then use check_password_hash() to check if
43             user = User.query.filter_by(email=email).first()
44             if user:
45                 if check_password_hash(user.password, password):
46                     flash( message: "Login in successfully.", category="success")
47                     login_user(user, remember=True)
48                     return redirect("authorizedHomepage") # change this later
49                 else:
50                     flash( message: "Password does not match. Try Again.", category="error")
51             else:
52                 flash( message: "Email does not exist.", category="error")
53
54     return render_template( template_name_or_list: "LoginPage.html", user=current_user )
55
```



# CONTROLLER/ ROUTING



# TEMPLATE INHERITANCE

```
PC  cs122_final_project - base.html
1  <!DOCTYPE html>
2  <html data-bs-theme="light" lang="en">
3
4  > <head...>
30
31  <body>
32  > <nav class="navbar navbar-expand bg-primary navigation-clean navbar-light" ...>
57
58  {% with messages = get_flashed_messages(with_categories=true) %} {% if
59  messages %} {% for category, message in messages %} {% if category =
60  'error' %}
61  > <div class="alert alert-danger alert-dismissible fade show" role="alert" ...>
62  {% else %}
63  > <div class="alert alert-success alert-dismissible fade show" role="alert" ...>
64  {% endif %} {% endfor %} {% endif %} {% endwith %}
65
66  {% block content %} {% endblock %}
67
68  > <footer class="text-center" ...>
69
70  <script src="/static/bootstrap/js/bootstrap.min.js"></script>
71  <script src="/static/js/bs-init.js"></script>
72  <script src="/static/js/Simple-Typing-Carousel-text-typing.js"></script>
73  ⚡ Optional JavaScript ⚡
74  ⚡ jQuery first, then Popper.js, then Bootstrap JS ⚡
75
76  > <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" ...>
77
78  > <script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js" ...>
79
80  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js"
81  integrity="sha384-JZR6Spejh4U02d8j0t6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
82  crossorigin="anonymous"></script>
83
84  </body>
85
86  </html>
```



```
{% extends 'base.html' %} {% block title %}Search{% endblock %}

{% block content %}
<div class="container d-md-flex d-xxl-flex justify-content-center justify-content-md-center justify-
<div class="col" style="...">
  <div class="row" style="...">
    {% for event in events %}
      <div class="col-4 py-3 mx-auto col-xl-4 col-lg-6 col-md-6 col-sm-12"
        style="...">
        <div class="card" style="...">
          <!-- Dynamically change the source of the image using Jinja2 -->
          
          <div class="card-body">
            <!-- Use Jinja2 to insert event details dynamically -->
            <h5>{{ event.event_name }}</h5>
            <p>{{ event.venue }}, {{ event.city }}</p>
            <p>{{ event.event_date }}</p>
          </div>
          <!-- Dynamically change the href to link to the Ticketmaster event page -->
          <div class="card-footer text-center">
            <small>
              <a href="{{ event.event_url }}" style="...">
                <i class="fa fa-eye pe-1"></i>View Event
              </a>
            </small>
          </div>
        </div>
      </div>
    {% endfor %}
  </div>
</div>
{% endblock %}
```

JINJA

```

@routes.route(rule="/authorizedHomepage", methods=["POST", "GET"])
@login_required
def authorizedHomepage():
    if request.method == "POST":
        keyword = request.form.get("search")
        if not keyword:
            return flash( message: "keyword cannot be empty.", category="error")
        # redirect(url_for('routes.get_event', keyword = keyword))
        event_list = []

        for i in range(
            2
        ): # you can change the number in range(num) for how many pages you want

            # get the json data
            event_data = fetch_event_details(Config.api_key, i, keyword)

            # Check if the response is an error message
            if isinstance(event_data, str):
                return make_response( *args: jsonify({"error": event_data}), 403)

            # get the events list in json file
            events = event_data.get("_embedded", {}).get("events", [])

            # create the dictionary for each event by traversing all the events
            for event in events:

                # create the new dictionary
                event_info = {}

                # get the event name, data, url
                event_info["event_name"] = event.get("name")
                event_info["event_date"] = (
                    event.get("dates", {})
                    .get("start", {})
                    .get("localDate", "Date not available")
                )
                event_info["event_url"] = event.get("url", "URL not available")

                # get the "venues" list for event
                venues = event.get("_embedded", {}).get("venues", [])

                # get the "city" information in the list
                for venue in venues:

                    # get the city name for the event and check the city is matched
                    city = venue.get("city", {}).get("name", "")
                    event_info["venue"] = venue.get("name", "Venue not available")
                    event_info["city"] = city

```

# API WEB SCRAPER



```

# get the image url
images = event.get("images", [])
if images and len(images) > 3:
    event_info["image_url"] = images[8].get(
        "url", "No image available"
    )

# if the city is matched, the dictionary is added to the list 'event_list'
if event_info.get("event_name") and event_info.get("venue"):
    event_list.append(event_info)

if len(event_list) == 0:
    flash(
        message: "There are no events happening in that area. Try again",
        category="error",
    )
else:
    return render_template(
        template_name_or_list: "SearchEventPage.html", events=event_list, user=current_user
    )

return render_template( template_name_or_list: "AuthorizedHomepage.html", user=current_user)

# get the api key and page number
1 usage  📌 Jonathan Manzano
def fetch_event_details(api_key, page, keyword):

    # url for events
    url = f"https://app.ticketmaster.com/discovery/v2/events.json?apikey={api_key}&locale=*page={page}&keyword={keyword}"

    # request
    response = requests.get(url)

    # return the json if the data is successfully retrieved.
    if response.status_code == 200:
        return response.json()
    else:
        return "Failed to retrieve events."

```



# CONCLUSION AND FUTURE DEVELOPMENT

EventRadar address the challenge of event discovery by providing a centralized platform that aggregates a wide range of events from Ticketmaster, secures user accounts with hashing algorithms, and offering a modular design that facilitates scalability.

## Future Improvements:

- Sentence Recommendation in the Search Bar
- Google OAuth Integration
- SERP API Integration