Robert Veloya
Hyun Woo Kim
Jonathan Manzano

TABLE OF CONTENTS

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

LIVE DEMO

cs122_final_project – app__init__.py

```python
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager
# from flask_migrate import Migrate
from app.config import Config
from os import path

# Create the database instance globally
db = SQLAlchemy()
DB_NAME = 'eventradar.db'
# Initialize login manager
login_manager = LoginManager()
# Login manager needs a view to redirect to when a login is required.
login_manager.login_view = 'auth.login'
# Initialize database migration tool
# migrate = Migrate()
```

FLASK

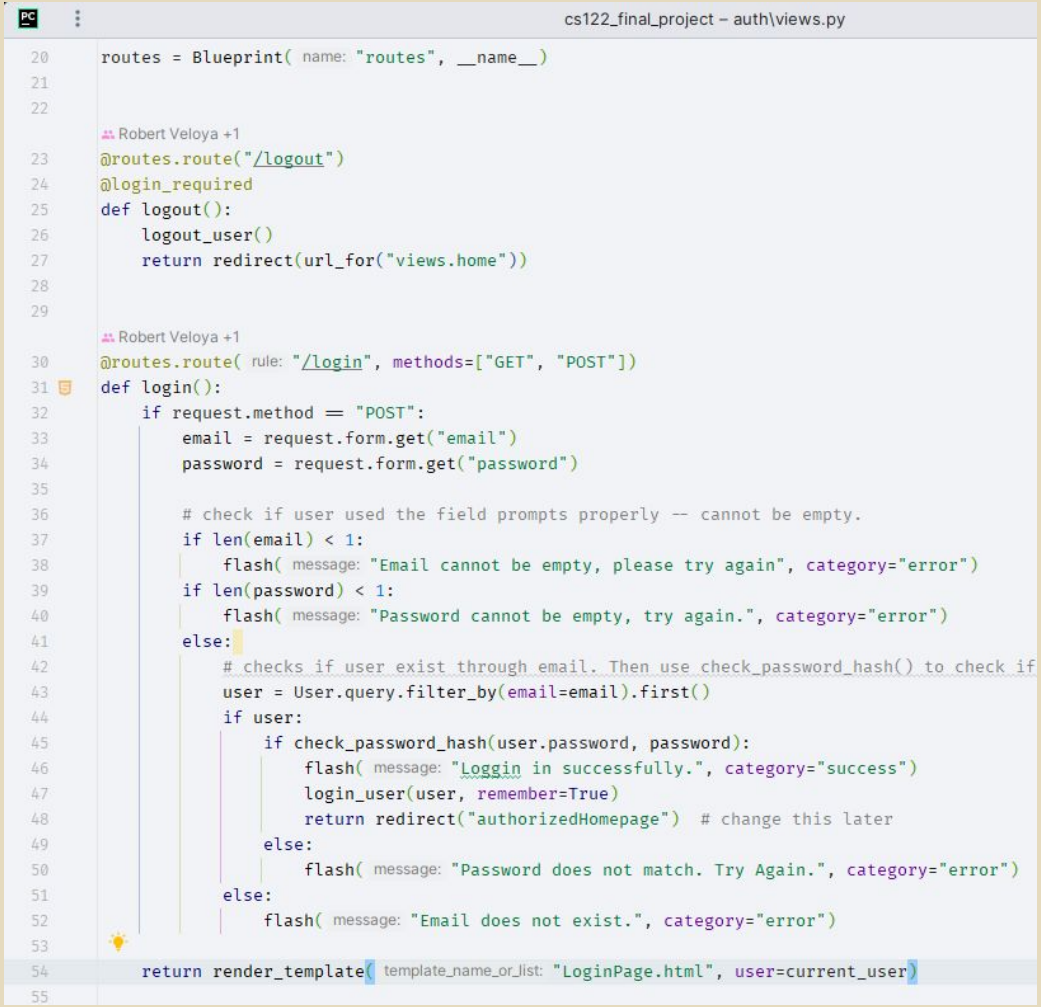
MODEL

cs122_final_project – models.py

```python
from datetime import datetime
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin
from werkzeug.security import generate_password_hash, check_password_hash
from app import db
#, login_manager

# User Loader for Flask-Login
# @login_manager.user_loader
# def load_user(user_id):
#     return User.query.get(int(user_id))

Robert Veloya
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), unique=True, nullable=False)
    # username = db.Column(db.String(64), index=True, unique=True)
    first_name = db.Column(db.String(20), index=True, nullable=False)
    password = db.Column(db.String(100), index=True, nullable=False)
    #relationship between user and event to store all events users sign up for.
    # event = db.relationship('Event')

Robert Veloya
class Event(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(128), nullable=False)
    description = db.Column(db.Text, nullable=True)
    location = db.Column(db.String(128), nullable=False)
    # 4/18/2024 @ 1:37 pm check if utcnow is causing any issues
    start_time = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    end_time = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    created_at = db.Column(db.DateTime(timezone=True), index=True, default=datetime.utcnow)
    # user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
```

Database

- eventradar 1
  - main
    - tables 3
      - event
      - sqlite_master
      - user
        - columns 4
        - keys 2
        - indexes 3
    - Server Objects



CONTROLLER / ROUTING

TEMPLATE INHERITANCE

cs122_final_project – base.html

```
<!DOCTYPE html>
<html data-bs-theme="light" lang="en">

<head...>

<body>
<nav class="navbar navbar-expand bg-primary navigation-clean navbar-light"...>

{% with messages = get_flashed_messages(with_categories=true) %} {% if
        messages %} {% for category, message in messages %} {% if category ==
        'error' %}
    <div class="alert alert-danger alert-dismissible fade show" role="alert"...>
{% else %}
    <div class="alert alert-success alert-dismissible fade show" role="alert"...>
{% endif %} {% endfor %} {% endif %} {% endwith %}

{% block content %} {% endblock %}


<footer class="text-center"...>
<script src="/static/bootstrap/js/bootstrap.min.js"></script>
<script src="/static/js/bs-init.js"></script>
<script src="/static/js/Simple-Typing-Carousel-text-typing.js"></script>
<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"...>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js"...>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js"
        integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
        crossorigin="anonymous"></script>
</body>

</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 -->
                        <img class="img-fluid card-img-top" style="..." src="{{ event.image_url }}">
                        <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

API WEB SCRAPER

```python
@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
```

```python
                    # 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 mateched, 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)
```

```python
# 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