EventGarden

Design Document

Emily McCaffrey (ekm57@pitt.edu), Allyson Morgan (amm946@pitt.edu), Abbey Kosmalski (alk320@pitt.edu)

Table of Contents:

1. Introduction

1.1 Purpose
Event Garden is an event management system for business people to create, manage, view, sign up, and search for various events, such as conferences, workshops, and meetups.

1.2 Scope
Event Garden was designed to provide those in the field of business in the Pittsburgh area with an environment in which they can reliably plan team meetings and events located at various venues throughout the city. The Event Garden website aims to provide users with a simple-to-use scheduling software in which it is easy to view past, current, and upcoming events. Those viewing the website can create a user account where they can view and sign up for events, while those looking to manage events can create an event manager account where they will be able to view, create, edit, and delete events.

---

2. System Overview
The system will utilize Object-Oriented Programming, Flask, and SQLite to manage users and events.

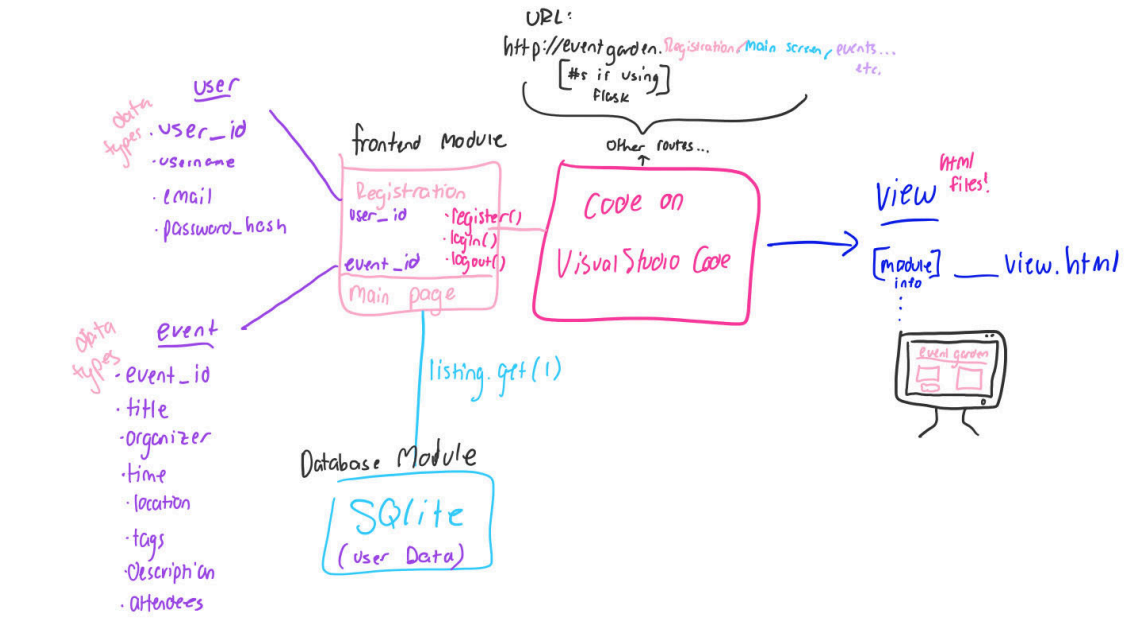
2.1 System Requirements We Intend to Include:
-Easy to navigate and aesthetically pleasing landing page
-Search function for upcoming events utilizing keywords
-All types of users will be able to view event details
-Login portal with username and password
-User profile page and settings page
-Account type for event managers
-Profile page for event managers
-Ability to archive past events
-Modern and user-friendly interface
-Supports a reasonable number of users
-Persistent storage
-Properly setup database
-Filter System for Events
-Event detail page will provide the map information of the event location and links that will create calendar events to Outlook with the option to download the calendar event.

---

3. System Architecture

3.1 Overview

- Frontend Module
    - The user interface where people interact with the system.
- Backend Module
    - The behind-the-scenes system that processes requests and makes things work.
- Database Module
    - Stores information about users and events.
- Authentication Module
    - Manages user login and registration securely.
- Event Management Module
    - Allows event organizers to create, update, and manage events.
- User Management Module
    - Keeps track of user profiles and event signups.



3.2 Subsystems & Responsibilities

Frontend Module: Main interface of Event Garden app
Shows events and user details.
Talks to the backend using API calls.

Backend Module:
Handles things like signing up for events and updating profiles.
Connects the frontend to the database.

Database Module: We're going to use SQLite
Stores information about users, events, and registrations.
Keeps the data accurate and consistent.

Authentication Module:
Handles logging in and signing up users.
Keeps passwords secure.

Event Management Module:
Lets event managers create, update, and delete events.
Keeps track of event details and who's attending.

User Management Module:
Manages user profiles and event history.
Lets users change their information.

3.3 Relationships Between Modules

- The frontend calls the backend
- The backend processes it and retrieves data from the database.
- The authentication module secures user interactions.
- The event and user management modules coordinate event-user interactions.

3.4 Object-Oriented Design

User Class

```
class User:
    def __init__(self, user_id, username, email, password_hash, role):
        self.user_id = user_id
        self.username = username
        self.email = email
        self.password_hash = password_hash
        self.role = role  # Regular User or Event Manager
```

Event Class

```
class Event:
    def __init__(self, event_id, title, organizer, time, location, tags, description):
```

```
        self.event_id = event_id
        self.title = title
        self.organizer = organizer
        self.time = time
        self.location = location
        self.tags = tags
        self.description = description
        self.attendees = []
```

EventManager Class (inherits from User)

```
class EventManager(User):
    def create_event(self, event):
        pass

    def edit_event(self, event_id, new_details):
        pass
```

3.5 API Endpoints

- User Authentication
    - POST /register → Register user
    - POST /login → Login
    - GET /profile → View profile

- Event Management
    - GET /events → View all events
    - POST /events → Create new event
    - PUT /events/{event_id} → Update event
    - DELETE /events/{event_id} → Cancel event

- User Event Interaction
    - POST /events/{event_id}/signup → Register for an event
    - DELETE /events/{event_id}/signup → Cancel signup

3.6 Justification for Design

This design is made to keep the code simple and organized. The frontend is where users interact with the system, and the backend handles all the behind-the-scenes work. The database stores important information like users and events. The authentication module makes sure users log in safely. The event and user management modules help organize and track events and user activities. Using API calls to connect the frontend and backend ensures that everything works smoothly and securely, making the system easy to use and simple.

---

4. Data Design

4.1 Data Description

The Event Management System organizes and processes data about users and events. It stores this information in a database and processes it through the backend to make the system work.

We're going to use SQLite to store the user data.

- User Data
  The system stores information like username, email, password, and user role (user or manager)
- Event Data
  Events are stored with details like the event title, time, location, tags, and description. The system allows event managers to create, edit, and delete events.
- Registration Data
  When a user signs up for an event, the system records which user registered for which event.

4.2 Dictionary

The system includes the following key data types:

- User – A person using the system.
  - user_id – Unique number to identify the user.
  - username – Name the user goes by.
  - email – User's email
  - password_hash – User's password.
  - role – Type of user (regular user or manager).
- Event – An event listed in the system
  - event_id – Unique number to identify the event.
  - title – Name of the event.
  - organizer – Person hosting the event.
  - time – Date and time of event
  - location – Where the event happens.
  - tags – Keywords that describe the event.
  - description – Details about the event.
  - attendees – List of users attending the event.
- Registration – A record of a user signing up for an event.
  - user_id – ID of the user attending.
  - event_id – ID of the event.

- Authentication – Handles user login and sign-up.
    - register() – Registers a new user.
    - login() – Logs in a user.
    - logout() – Logs out a user.

- EventManager – A user who can create and manage events.
    - create_event() – Creates a new event.
    - edit_event() – Updates event details.
    - delete_event() – Removes an event.

---

5. Human Interface Design

Users will be able to easily navigate the system using a mouse and keyboard. The login page will allow users to log in as regular users, managers, or register a new account. Once logged in, users will be able to access the system. It will automatically take them to the user dashboard, which will display the events they are currently signed up for, events they may be interested in, an option to edit their profile, and a search bar and filters to search for new events.. For event managers, their dashboard will display the events they are currently managing with the ability to edit and view the details of their event, and the events created by other managers. They also have the ability to view an archive of all past events. Below is a very loose, preliminary sketch of the user interface design (subject to change).

Python Final Project