B. Tech. Industrial Training Report

On

Affairs To Remember

BY

VINAY YADAV(17EEBCS303)

DEPARTMENT OF COMPUTER SCIENCE
ENGINEERING COLLEGE BIKANER

December 2019

Python Technology

INDUSTRIAL TRAINING REPORT

Submitted in partial fulfillment of the
requirements for the award of the degree

of
BACHELOR OF TECHNOLOGY
in

Computer Science Engineering

Submitted by:
VINAY YADAV(17EEBCS303)

Guided by:
Dr. Lalit Malotiya
Assistant Professor

Submitted to:

Mr. Prashant Yadav
Assistant Professor

ENGINEERING COLLEGE BIKANER

CANDIDATES DECLARATION

We hereby declare that the Industrial Training Report entitled "AFFAIRS TO REMEMBER" submitted in partial fulfillment for the award of the degree of Bachelor of Technology in "Computer Science Engineering" completed under the supervision of Dr. Lalit Malotiya Assistant professor in Python technology of C-DAC ATC NETCOM, JAIPUR is an authentic .

This work is further supervised by Mr. Prashant Yadav Assistant professor in IT department of Engineering college Bikaner

Further, I/we declare that I/we have not submitted this work for the award of any other degree elsewhere.

Vinay Yadav

(Mr. Prashant Yadav)

(Assistant Prof. IT DEPTT.)

Mr. Maninder Singh Nehra

(HOD of CSE Department)

CERTIFICATE by Industrial Training Guide

TABLE OF CONTENTS

ABSTRACT

Event management, the very topic looks challenging. A concept which gained importance in India only after the late 90's. Commitment, leadership and mental & physical devotion are the core factors needed to manage any type of event. Irrespective of the type or the scale of the event, the mental and physical hard work that is to be put in, differs by only a negligible degree of difference.

This terminology is comparatively very new to India, though Indians have been arranging for wedding ceremonies, naming & threading ceremonies even much before independence. But due to the lack of proper forecasting, proper material handling they used to end up in problems like wastage of the food due to less people coming in or fire in the pandal or food poisoning .These problems many a times used to put the families into financial trouble after the wedding.

It is very easy for the audiences to make the event a hit or a flop. It takes just the 5 minutes for the audiences to judge the event resulting in the efforts of nearly 3-4 months and the hard work 70-80 people either turning productive or waste. Thus the efforts they have put in always remains at stake till the date of the event.

There are innumerable activities that have to be carried out. First of all forming

committees, then allotting different jobs to each committee is the very first step. Here all the theoretical concepts learnt up till now in subjects like public relations, human resource planning, logistics, human skills, controlling,accounts, organizing, and others come into actual use.

As an event manager one must have a lot of flexibility in terms of working pattern. Be free to do all sorts of jobs irrespective of your position.

1.INTRODUCTION

1.1 Project Definition

This project works on the basic fundamentals of Python and Django. Event Managemnet is a process of evaluating and analyzing a Event and then working on making those events beautiful and memorable.Event management is a process of organizing a professional and focused event, for a particular target audience. Affairs to Remember would mostly aim at the social and commercial events that needs proper organization and coordination of different aspects of an event.Here in "Affairs To Remember" we work to provide best experience at minimum cost.We create a medium for user to let them enjoy there moments without any stress.

1.2 Scope

The name of the application is “Affairs To Remember”. A web application of the same already exists but we have combined to different applications in one . Users can select the venue with time and date from our site. The application has two parts to it, the Dinner booking and party booking. The venue theme and date is selected by the users ,date will get allotted if not already been taken . Booked history will be available for the users and it can be seen by there registration number.

1.3 Objective

The main objective to create this application is to bring about a different product in the market the combination of to existing products to increase the competitive nature. Zomato and Event Organizer sites are the already existing applications used to book dinner tables at restaurants and venue for parties . But no one is providing it together at one place.

This application aims at providing the platform for the organizing both dinner and parties at an one place and at affordable range.

1.4 Tools & Technology Used

- PYCHARM (Coding)
- SQLITE (Database)

2.0 Project Management

2.1 Project Planning

The project is approximately calculated to complete within a period of 4 months. First, the layout design was discussed along with the platform on which is to be worked upon. The study of Python Technology and its execution took about a month to complete then it was preceded with the coding and design portion of the project. At the end after careful evaluation, the project was termed successful.

2.1.1 Project Development Approach and Justification

The approach used to develop this application is agile scrum based methodology. Scrum is part of the Agile movement. Agile is a response to the failure of the dominant software development project management paradigms (including waterfall) and borrows many principles from lean manufacturing. In 2001, 17 pioneers of similar methods met at the Snowbird Ski Resort in Utah and wrote the Agile Manifesto, a declaration of four values and twelve principles. These values and principles stand in stark contrast to the traditional Project Manager's Body of Knowledge (PMBOK). The Agile Manifesto placed a new emphasis on communication and collaboration, functioning software, team self-organization, and the flexibility to adapt to emerging business realities. Scrum's early advocates were inspired by empirical inspect and adapt feedback loops to cope with complexity and risk. Scrum emphasizes decision making from real-world results rather than speculation. Time is divided into short work cadences, known as sprints, typically one week or two weeks long. The product is kept in a potentially shippable (properly integrated and tested) state at all times. At the end of each sprint, stakeholders and team members meet to see a demonstrated potentially shippable product increment and plan its next steps.

Scrum is a simple set of roles, responsibilities, and meetings that never change. By removing unnecessary unpredictability, we're better able to cope with the necessary unpredictability of continuous discovery and learning.

2.1.2 Project Effort and Time, Cost Estimation

Now we will calculate Effort, Cost and Project Duration Estimation using The Basic COCOMO Model which is itself a Heuristic Estimation Technique. As our project is an Application Development, so it will come under the Organic Category.

Estimation of Line of Code:

Line of code = no. of files * avg. no. of line

= 100 * 75

= 7500 =

$7.5\left(\text{KLOC}\right)$

Estimation of Development Effort:

Effort= $2.4\left(\text{KLOC}\right)1.05$ PM

= $2.4\left(7.5\right)1.05$ PM

= $2.4\left(8.3\right)$ PM

= 19.90 PM

= 20 PM

Estimation of Development Time:

Tdev= $2.5\left(\text{Effort}\right)0.38$ Months

= $2.5\left(20\right)0.38$ Months

= $2.5\left(3.12\right)$ Months

= 7.80 Months = 8 Months

2.2 Project Work Scheduling

	Task Name	Start Date	End Date	Duration
1	Requirement Analysis	06/01/19	06/16/19	10d
2	Design	06/16/19	06/26/19	10d
3	Implementation	06/26/19	07/20/19	24d
4	Testing	07/20/19	07/30/19	10d
5	Deployment	07/30/19	08/05/19	6d
6	Maintenance	08/05/19	08/09/19	4d
7				
8				

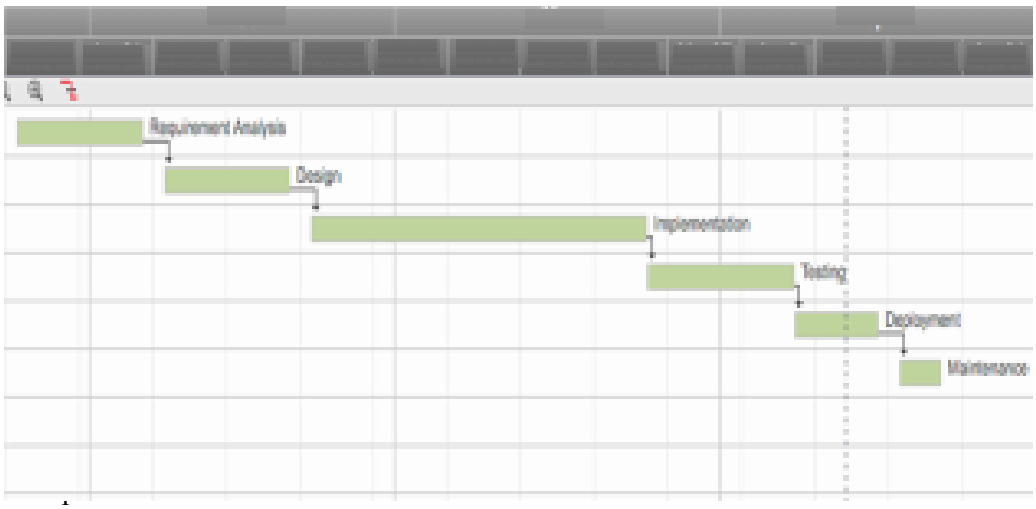

Fig 2.1 Gantt chart

3.0 System Requirements Study

3.1 User Characteristics

Users

- o The user can view different filter by city,veg ,non-veg and drinks.
- o Users don't have to create any account the registration number will be the account name or number itself .
- o They can venue can't be booked if it is booked on that date.
- o They can see the portfolio of the restaurants by clicking on the portfolio button .
- o The users can see there booking using there registration number.
- o In case of error regarding dates , users can go to their booked history on the application for the verification.

VENUE OWNERS

- o The owner can upload there images and there speciality.
- o They need to have an account on the site.
- o They orders will be taken after approaching them.

Administrator

- o The administrator has the full-fledged rights over the application.
- o Can add/delete venues.
- o Can view the accounts and events.
- o Can change the password.
- o Can hide any kind of features from the both of users and event organizers.
- o Insert/delete/edit the information of available on application.
- o Can access all the accounts of the users/owners of venue.

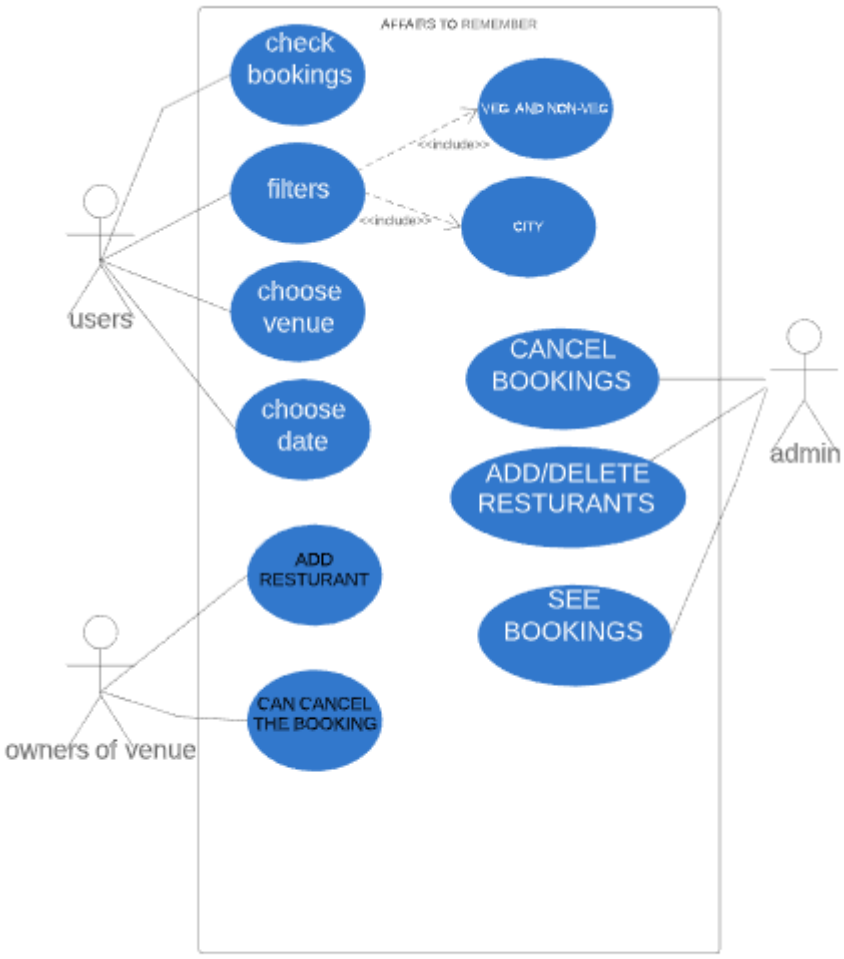

Fig 3.1 Use Case Diagram

3.2 Hardware and Software Requirements

FRONTED: PYCHARM

Pycharm is an integrated development environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django as well as Data Science with Anaconda.

- Coding assistance and analysis, with code completion, syntax and error highlighting, linter integration, and quick fixes
- Project and code navigation: specialized project views, file structure views and quick jumping between files, classes, methods and usages
- Python refactoring: includes rename, extract method, introduce variable, introduce constant, pull up, push down and others
- Support for web frameworks: Django, web2py and Flask
- Integrated Python debugger
- Integrated unit testing, with line-by-line code coverage
- Google App Engine Python development
- Version control integration: unified user interface for Mercurial, Git, Subversion, Perforce and CVS with change lists and merge

BACKEND: SQLITE

SQLite is a relational database management system contained in a C programming library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.

SQLite is ACID-compliant and implements most of the SQL standard, using a dynamically and weakly typed SQL syntax that does not guarantee the domain integrity.

SQLite is a popular choice as embedded database software for local/client storage in application software such as web browsers. It is arguably the most widely deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems, among others. SQLite has bindings to many programming languages.

JSON Restful Web services

JavaScript Object Notation is an open standard format that uses human text to transmit data objects consisting of attribute–value pairs. It is the most common data format used for asynchronous browser/server communication $\left(\text{AJAJ}\right)$, largely replacing XML which is used by AJAX. JSON is a language-independent data format. It derived from JavaScript, but now code to generate and parse JSON-format data is available in many programming languages. The official Internet media type for JSON is application/json.

4.0 System Analysis

4.1 Study of Existing System

Zomato is an Indian restaurant aggregator and food delivery start up founded in 2008. It was started by Deepinder Goyal and Pankaj Chaddah. It provides information, menus and user-reviews of restaurants, and also has food delivery options from partner restaurants in select cities. As of 2016, the service is available in 24 countries.

4.2 Limitations of Existing System

Zomato has not added the feature of selecting venues for parties. “Affairs To Remember” application aims at providing both the platform of dinner and party venue select for the users at an affordable range.By this we are giving a competition to Zomato and When there is no competition there is no platform for growth.

4.3 Requirements of Proposed System

4.3.1 Functional Requirements

- The application is an alternate option for the users to view and book events and venues.
- Only authorized person can access related details.
- The users will register themselves on the application by there phone numbers during the booking time. Users can change their information regarding the events.
- Qrganizers can see the bookings by there numbers.
- Administrator will be responsible for the overall maintenance of the application.

- The dates willbe confermed as per there allotation .

4.3.2 Non Functional Requirements

Before Application Development: ?

- Justification for Application Development.
    - o Describe the problem and how the application is useful to solve that problem.
    - o The app's focus has a strong connection to the target people.
    - o Is it innovative or already available in store? If available then,

      how it is different than available? ?

- Justification for the tools and technology used for the Application Development.
    - o Comparison of different technology available for application development

      and justification for your choice.
    - o Specification of tools and technology used for your application development. ?

- Justification for Hardware and Device selection.
    - o Justification and specification of hardware selection like, processor, RAM, OS version, etc.
    - o List of compatible devices on which the application will run.

At the time of Application Development: ?

- Aluto adjustment of GUI.

- o Usage of proper Layouts for GUI making. It should be in auto adjust GUI for different cell phone with different OS and also for different tablets.
- o App offers flexibility to alter content and settings to meet people's needs.

- Coding convention.
  - o Structure of the Application.
    - Maintain images in all four packages.
    - Maintain .manifest file, .java files, and .xml files properly.
  - o Naming Conventions. (Class, Objects, Methods, Variables, etc.)
  - o Proper comments (e.g. description of method in comment.)
  - o Study of proper License of Agreements.

- Application should be multithreaded.

- Proper Notification for process/task.
  - o Proper notification/Confirmation at the time of completion of any task or process.
  - o Progress-bar while loading.
- All necessary validation.
  - o Email Id validation.
  - o Password validation.
- Email Verification.
- Auto-update while connect with internet.
- Feedback functionality is necessary.

After Application Development:

- Application file size should be as minimum as possible. (<50Mb)
- Easy installation and Easy availability of application.
  - o Application should be easily installable in provided list of devices.
  - o Application should be available in store.
- List of different test cases.
  - o Application should run on different cell phone with different OS versions and at least one tablet.

4.4 System Work Flow

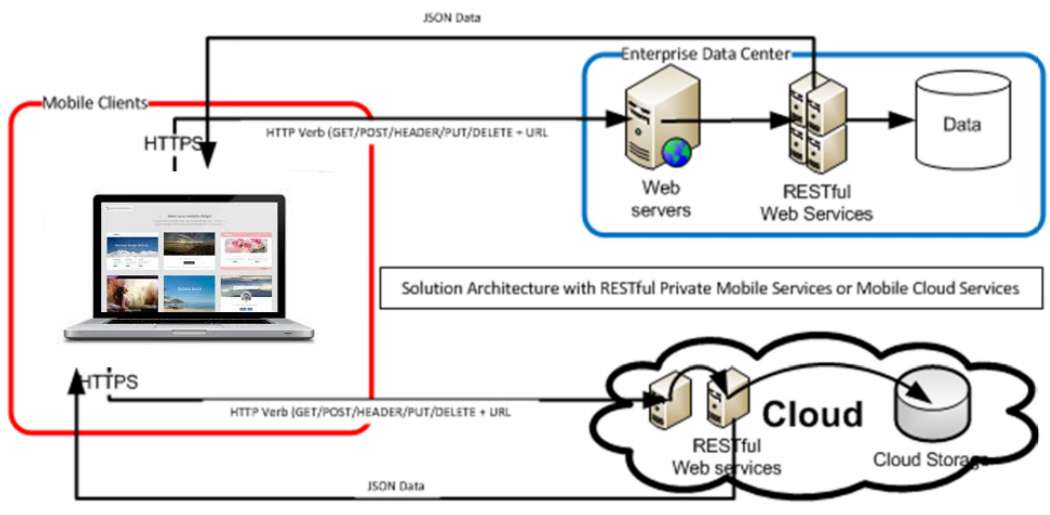

Fig. 4.1 System Workflow

4.5 Class Diagram

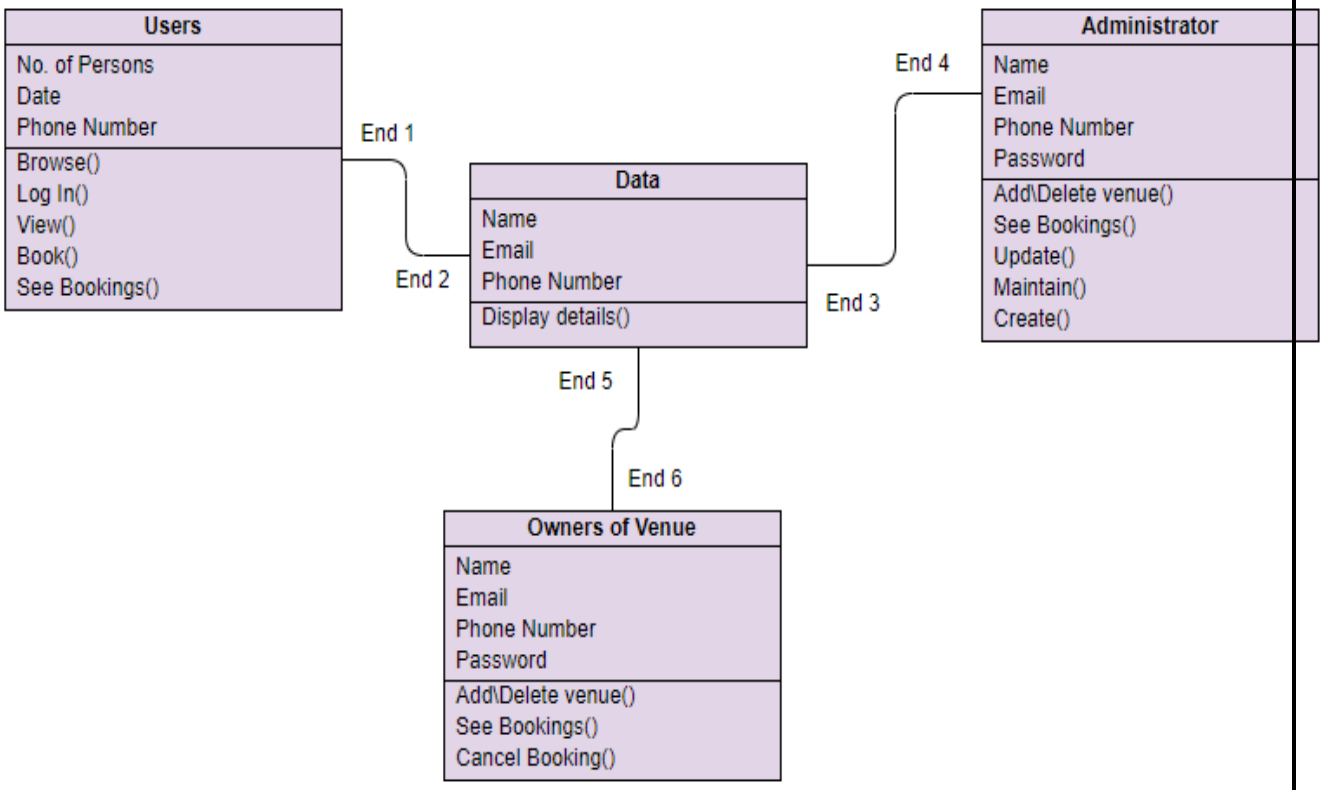

Fig 4.2 Class Diagram

4.6 Activity Diagram

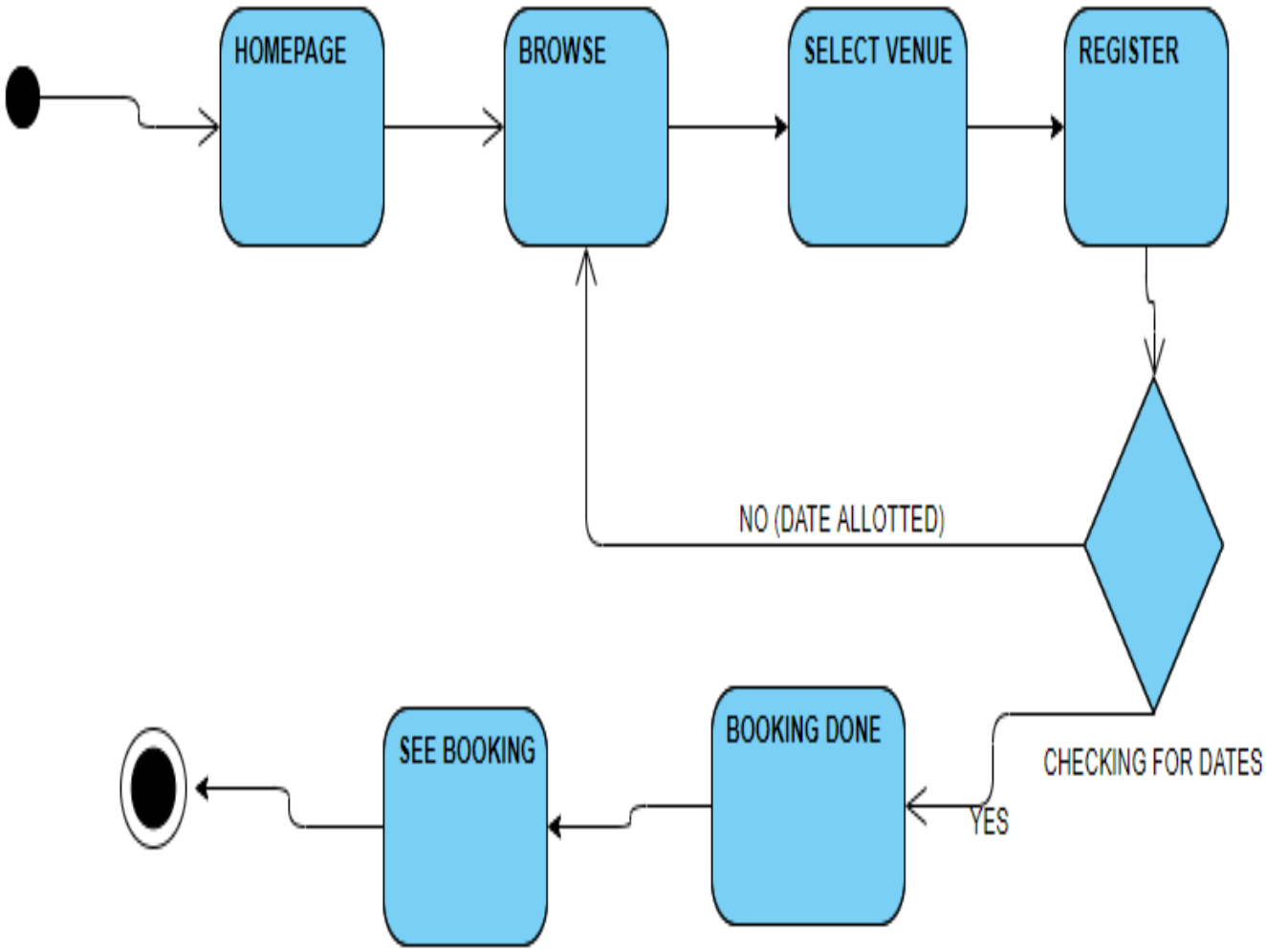

Fig. 4.3.1 User Activity Diagram

4.7 State Chart Diagram

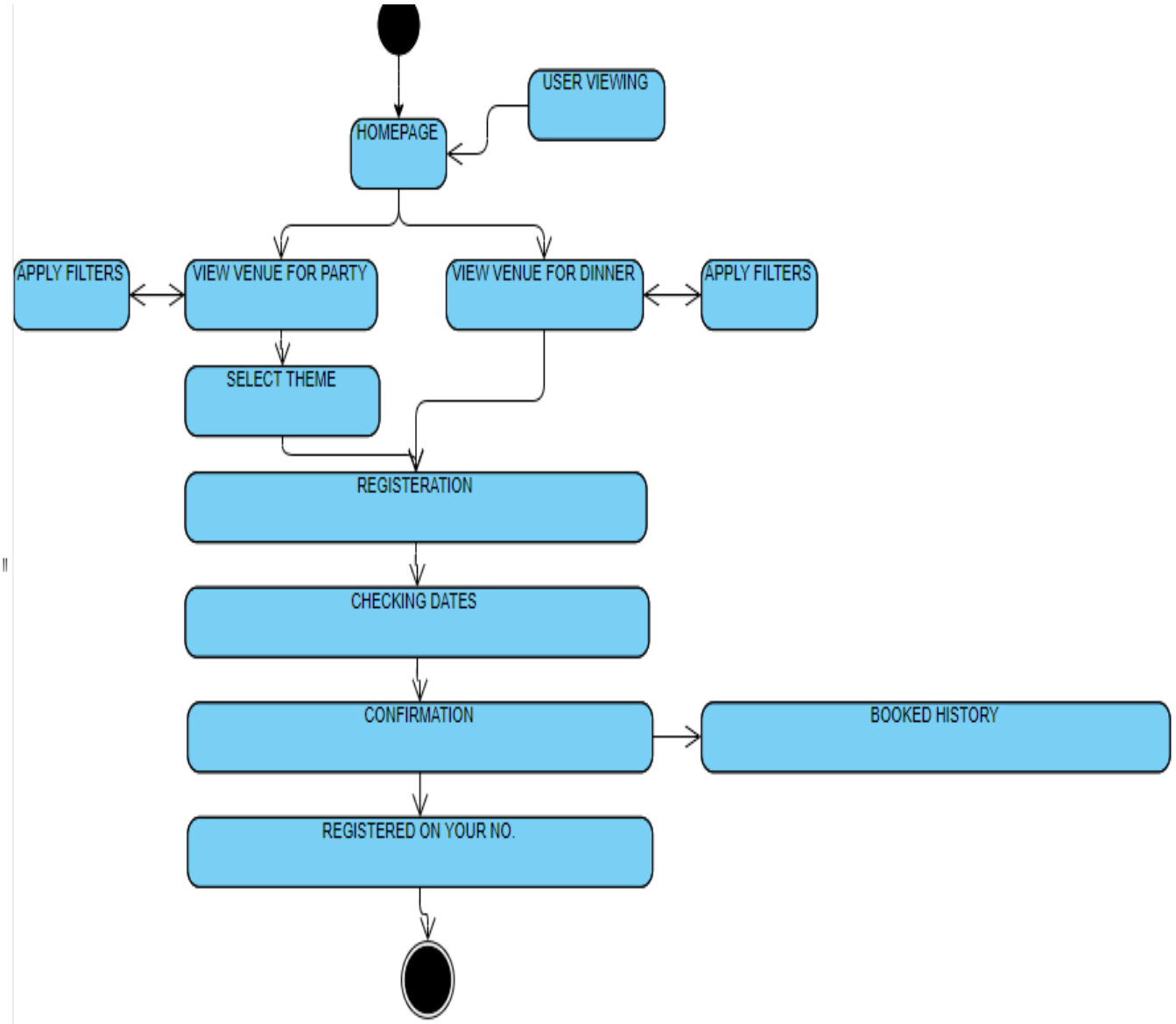

Fig. 4.5 State Chart Diagram

5.0 System Design

5.1 Homepage layout

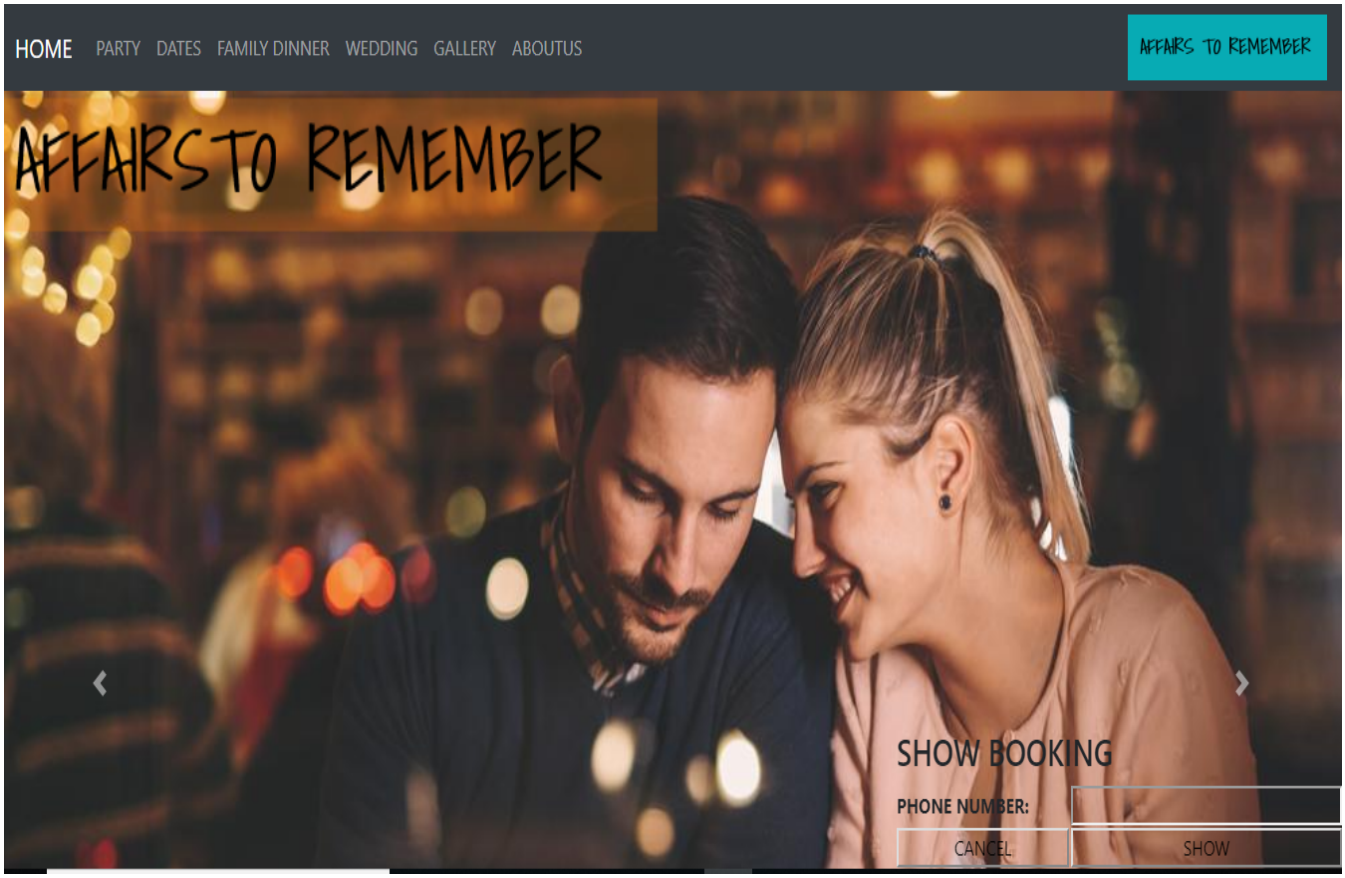

Fig. 5.1 Homepage

Homepage of “Affairs to remember” with show booking option

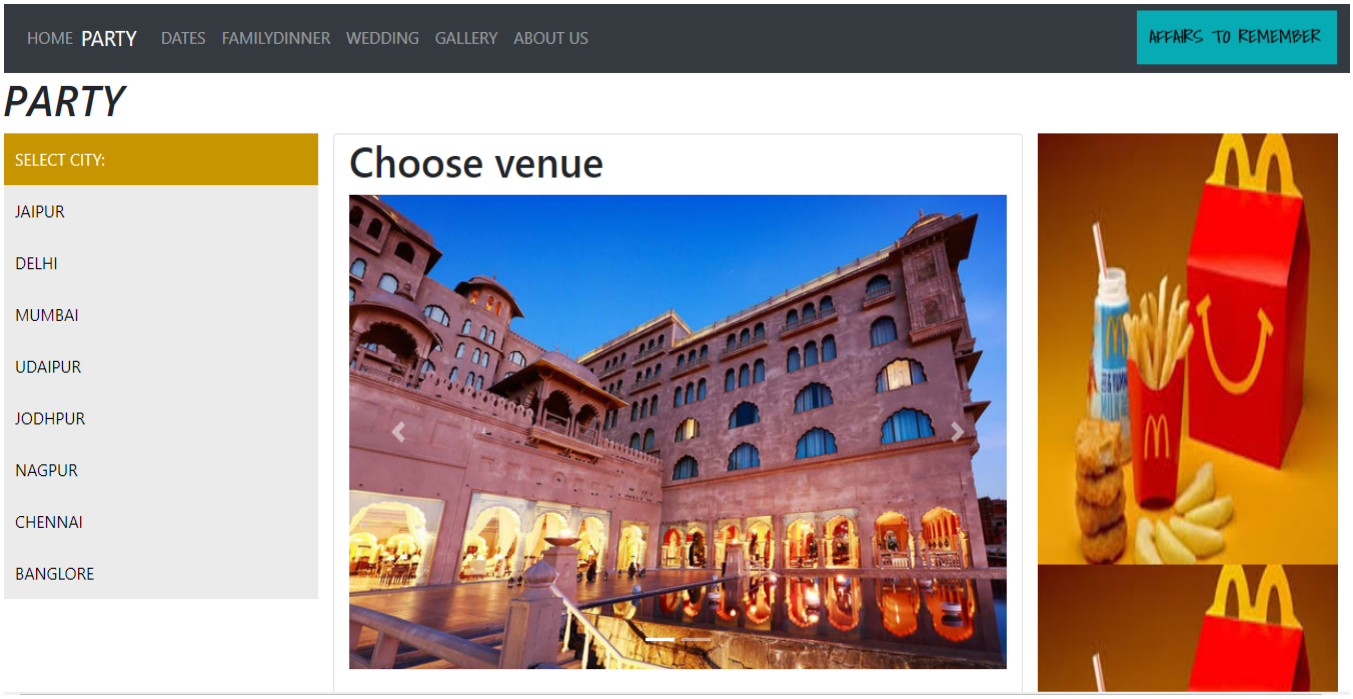

Fig. 5.2 Party Page

The party page with its city filters which is available to the users

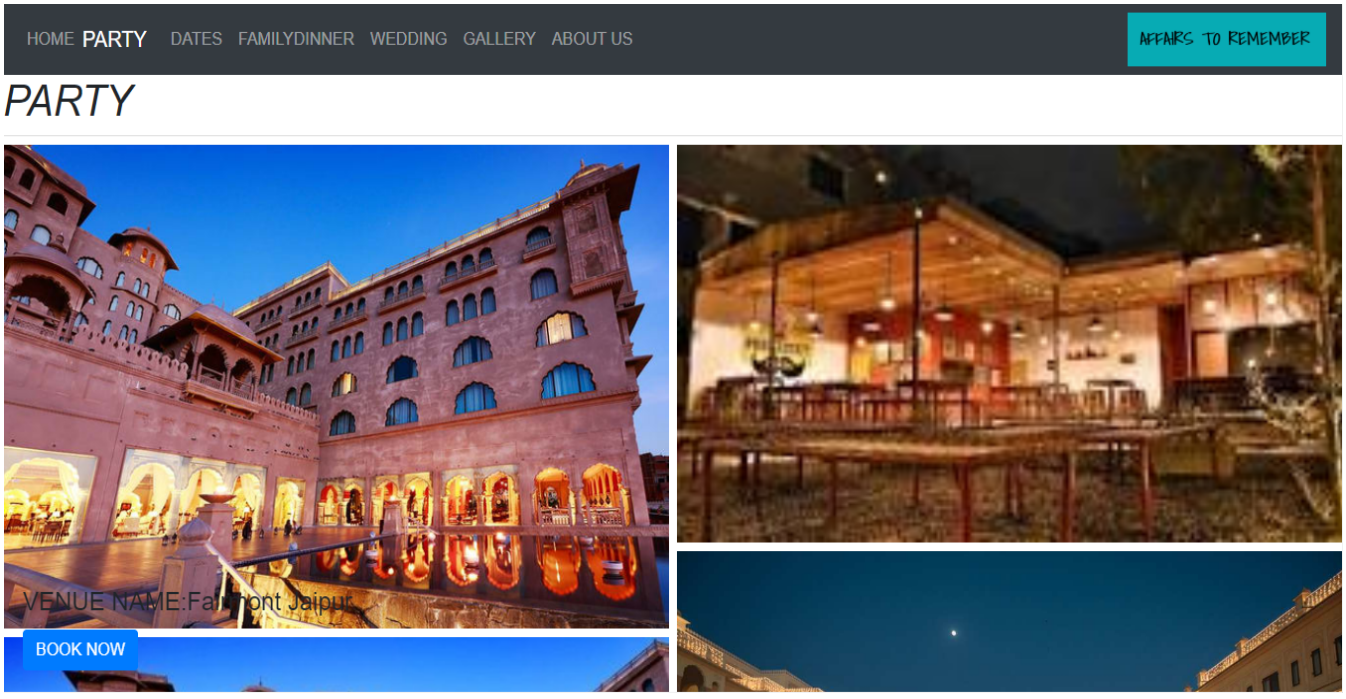

Fig 5.2.1 Portfolio page

The party portfolio page ,so that users can see more pics of the venue.

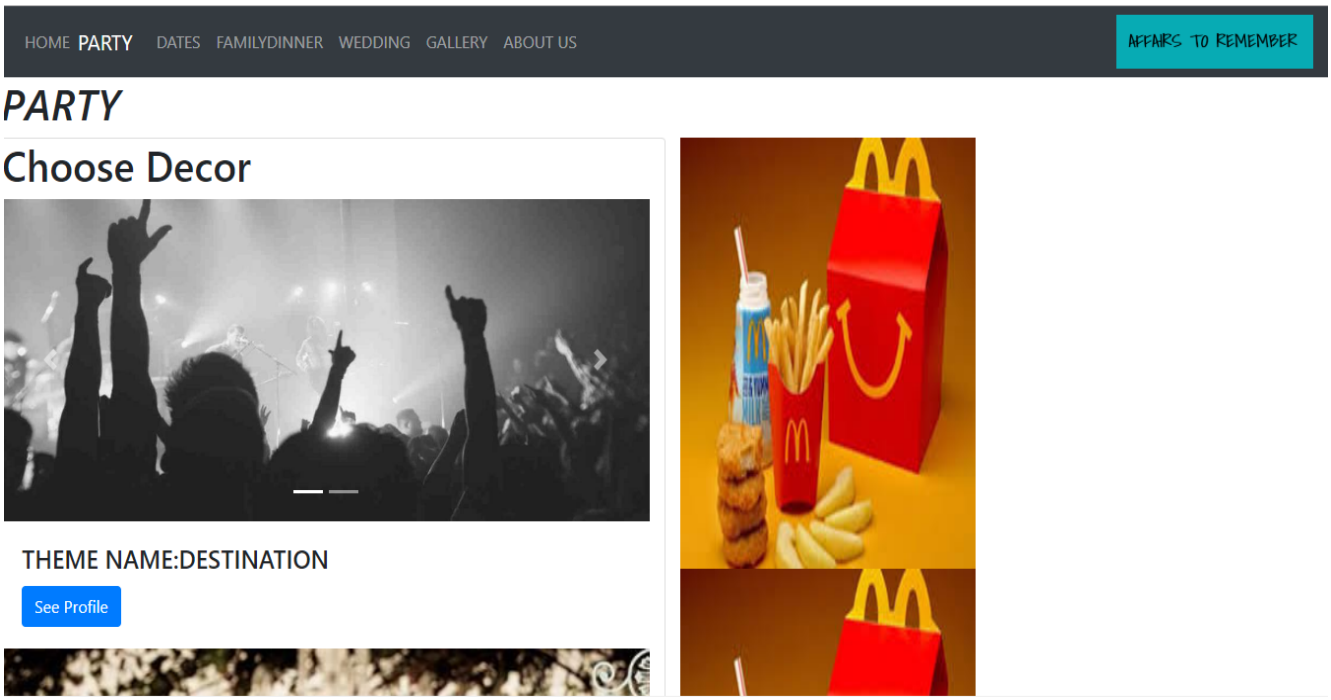

Fig 5.2.2 decor page

After selecting the venue for the party decor page is there to select the decoration of the party.

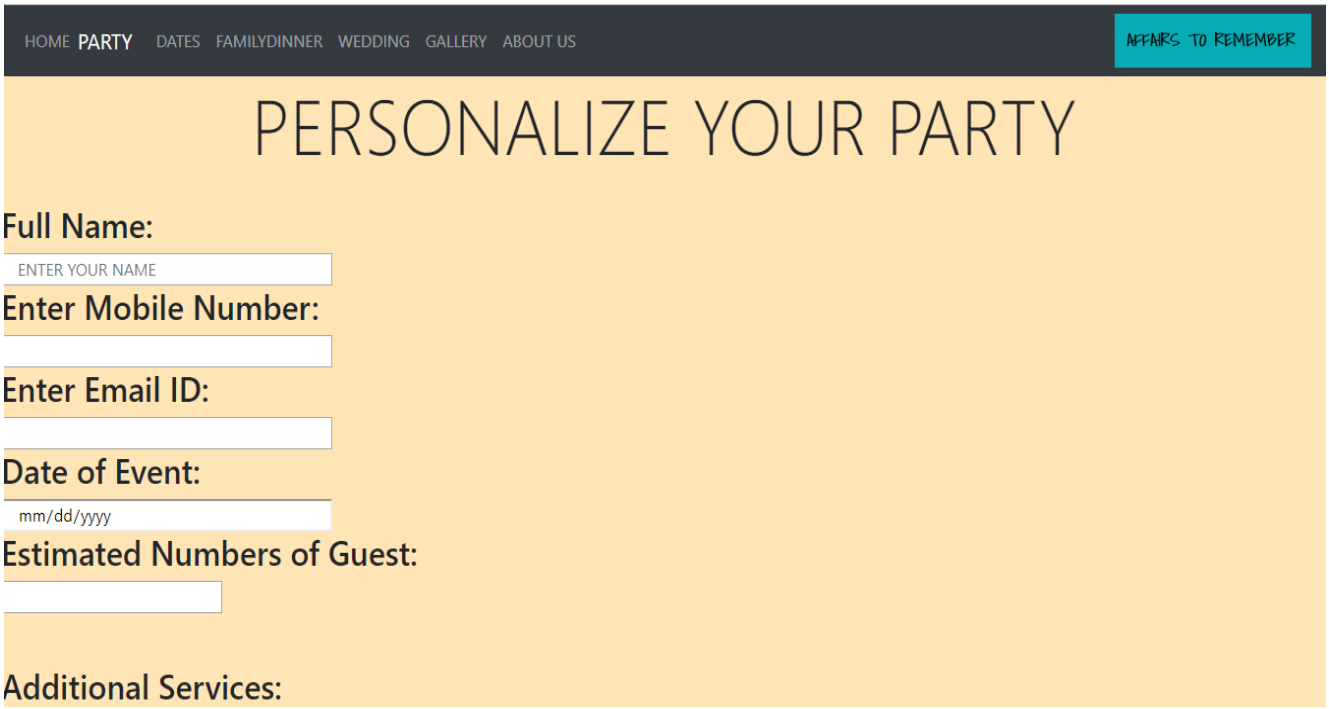

Fig 5.2.3 party registration page

Registration page for parties.

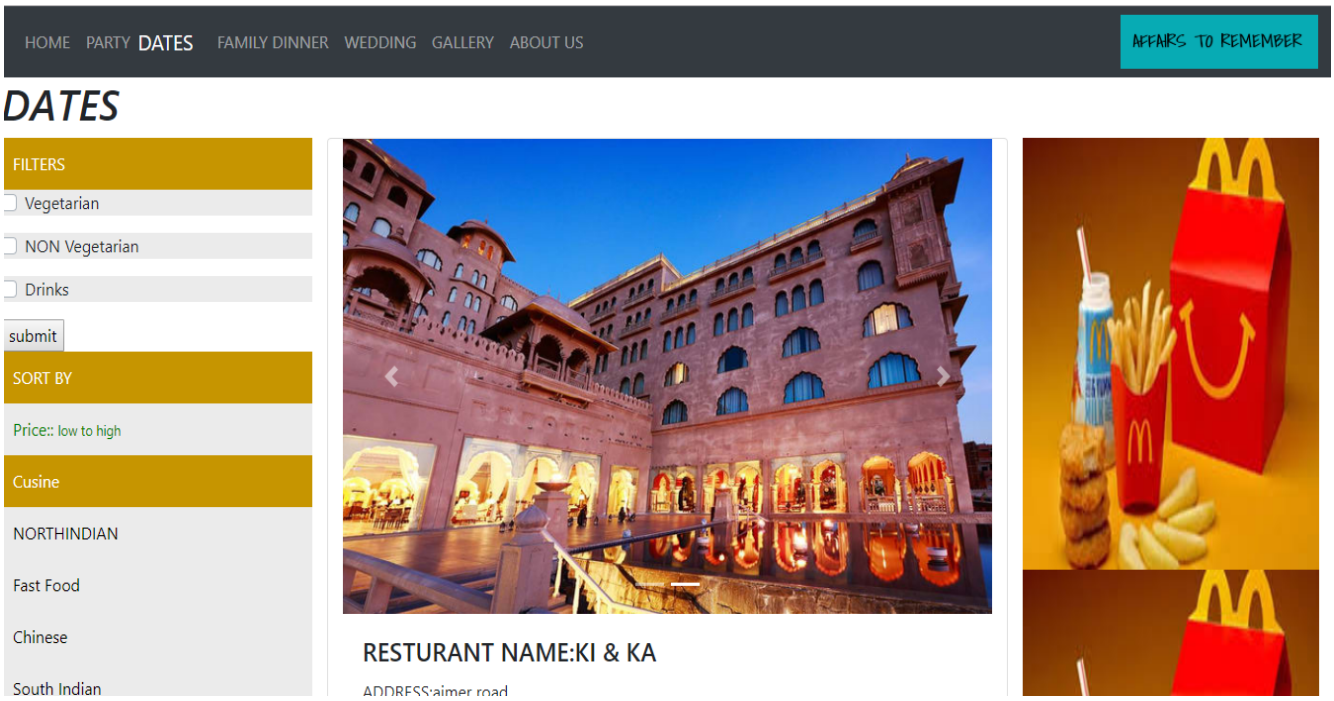

Fig 5.3 Dates\dinner page

This page have more filters then party page ,users can select its food choice with city.

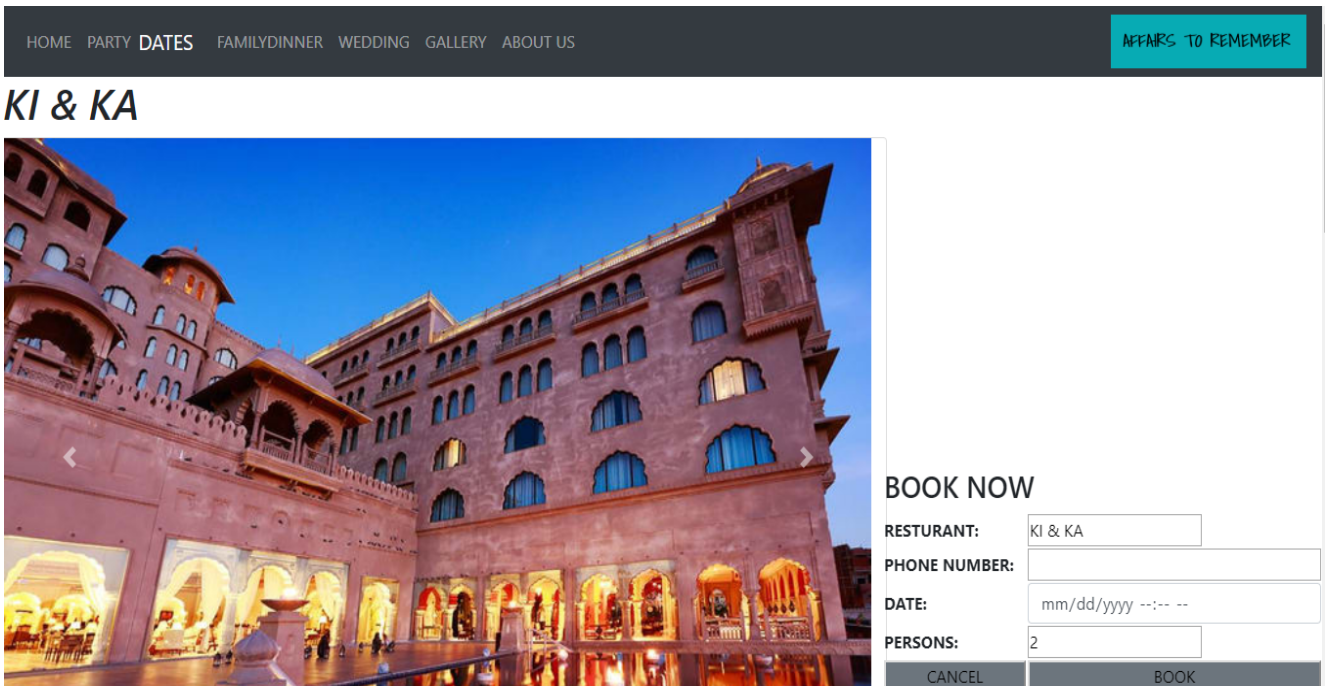

Fig 5.3.1 dates \dinner registration and detail page

Shows the details of the restaurant and register your table .

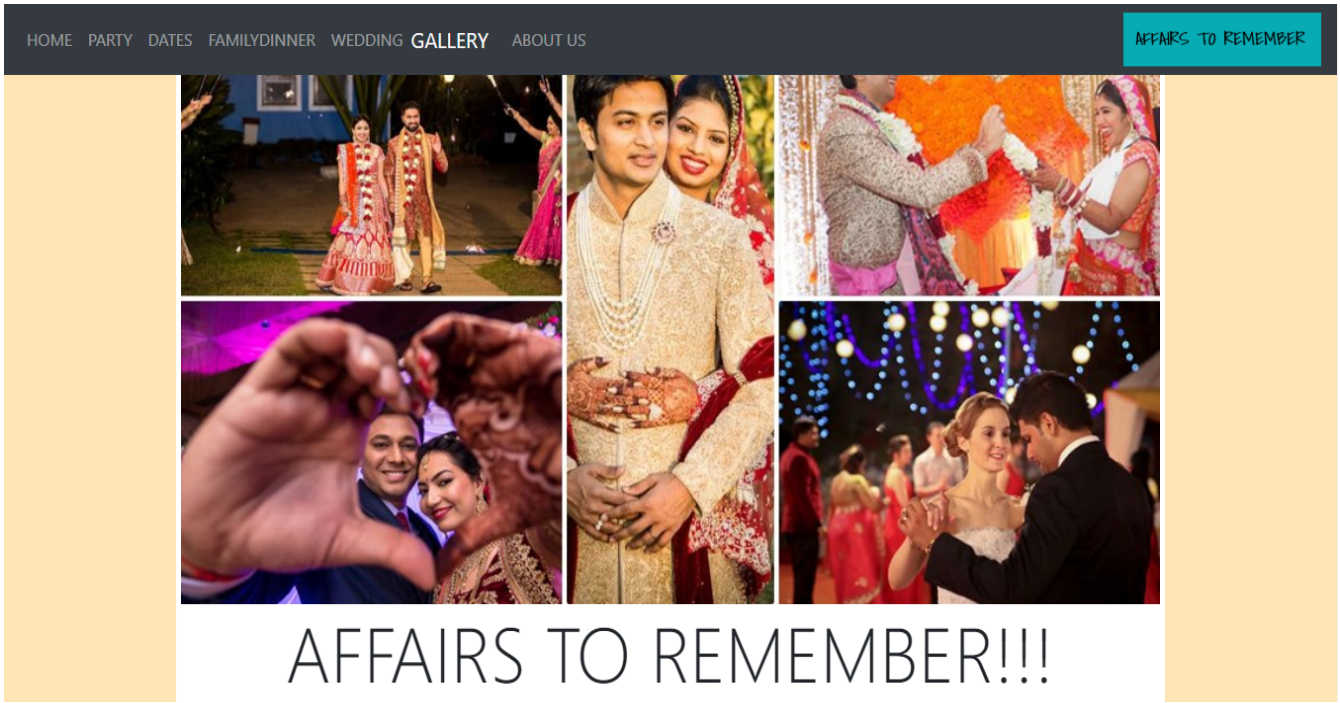

Fig 5.4 Gallery page of affairs to remembe

All the photos of all event done by “Affairs To Remember”

6.0 System Implementation and Testing

6.1 Coding Standards

- Write short methods – No official limit, but try to keep methods short and focused. Think often about how to refractor your code to break it into smaller and more reusable pieces. This is good advice in any language. ?
- This also shows why overly strict rules on the length of comments can be counterproductive by encouraging developers to write long methods to avoid writing docs. Keep lines short – They have a strict rule of 100 characters except for imports or comments that contain URLs or commands that cannot be broken up. Not sure 100 is the magic number, but short lines are good practice anyhow. ?
- Use JavaDoc from the beginning – Don't wait until the code is finished. Short comments are fine, but use some. Explain purpose and non-obvious behavior. Don't explain standard Java constructs. Document every class ?
- /** Represents a collection of Blahs. ?
- Used to ... **/ public class Foo { ... } ?
- Document anything public – Methods – Constructors – Instance variables (but very rare to have public ones)

• Use @Override when you override methods from parent class

– Won't be caught until run time

- Public void oncreate(Bundle savedInstanceState) { ... }

  – Will be caught at compile time

  @Override public void oncreate(Bundle savedInstanceState)

  { ... }

- Avoid empty catch blocks – Yes try { ... } catch(SomeException se) { doSomethingReal(); } – No try { ... } catch(SomeException se) { } – Marty's opinion • Usually, but not always, a good rule

- List each Exception type – Yes try { ... } catch(ExceptionType1 et1) { ... } catch(ExceptionType2 et2) { ... } – No try { ... } catch(Exception e) { ... }

6.2 Testing Methods

Monkey Runner is the tool used for testing. The monkey runner tool provides an API for writing programs that control an Android device or emulator from outside of Android code. With monkey runner, you can write a Python program that installs an Android application or test package, runs it, sends keystrokes to it, takes screenshots of its user interface, and stores screenshots on the workstation. The monkey runner tool is primarily designed to test

applications and devices at the functional/framework level and for running unit test suites, but you are free to use it for other purposes.

The monkey runner tool is not related to the UI/Application Exerciser Monkey, also known as the monkey tool. The monkey tool runs in an adb shell directly on the device or emulator and generates pseudo-random streams of user and system events. In comparison, the monkey runner tool controls devices and emulators from a workstation by sending specific commands and events from an API.

Argument	Description
`-plugin <plugin_jar>`	(Optional) Specifies a `.jar` file containing a plugin for monkeyrunner. To learn more about monkeyrunner plugins, see Extending monkeyrunner with plugins. To specify more than one file, include the argument multiple times.
`<program_filename>`	If you provide this argument, the `monkeyrunner` command runs the contents of the file as a Python program. If the argument is not provided, the command starts an interactive session.
`<program_options>`	(Optional) Flags and arguments for the program in <program_file>.

Table 6.1 Flags and Exceptions

7.0 Future Enhancement

The application has been designed at the maximum possible excellence. Still we accept drawbacks, as it is a human effort. As of now, the application is working as per requirements. The functional part of the application is lacking as we don,t have that much of knowledge to proceed further.

8.0 Conclusion

8.1 Self Analysis of Project Viabilities

Technology Issues	Market Issues
• Performance • Ease of learning • Ease of deployment • Ease of support • Operational characteristics (i.e. can it run 7 days a week, 24 hours a day?) • Interoperability with your other key technologies • Scalability	• Vendor viability (i.e. is it likely that they will be in business in two years? In five?) • Alternate sources for the technology, if any • Third-party support for related products and services • Level of support provided by the vendor • Industry mindshare of the product (i.e. is the market gravitating toward or away from this technology?)

Table 8.1 Technical Feasibility of Mobile Application.

8.2 Summary of Project work

In this project we worked on Django framework , where we design our website with the help of html and bootstrap and processing it with python. We have designed mainly six web pages including: home page, Party , Dinner , Weddings , Dates , Galleries ,About Us and an admin panel for the admin to handle the data and make changes . We have stored all the data given by the customer in a database and showcased it on the admin panel. Costumer can access and see its booking just by entering their details .It is a quite interactive site where you can customize your events, and we will reach you out for better experience.

References

1) https://edureka.co/blog/django-tutorial/

2) TheNewBoston:https://www.youtube.com/playlist?list=PL6gx4Cwl9DGBsvRxJJOzG4r4k_zLKrn xl

3) http://w3schools.com/

4) Software Requirements Specification for Problem Based Learning Module, Souman Mandal, 2010.

5) Software Design Specification (SDS) Acropolis Course Management System, 2011

6) IEEE Recommended Practice for Software Requirements Specifications, Software Engineering Standards Committee of the IEEE Computer Society. 1998

7) Software Requirements Specification for PPDP Contact Management System (CMS)

8) http://www.ehow.com/facts_5156877_preface-book.html, Sat. 29/10/2011.

9) http://www.sil.org/lingualinks/literacy/referencematerials/glossaryofliteracyterms/WhatIsAPreface.htm, Sat. 29/10/2011.

A.COMPLETE CONTRIBUTARY SOURCE CODE

1.Homepage.py

```
def homepage(request):
  success=False
  if request.method=='GET':
   return render(request,'homepage.html')
  if request.method=='POST':
      phone=request.POST.get('phone')
      user=book.objects.all().filter(phone=phone)
      user1=partyreg.objects.all().filter(phone=phone)
      success=True
      return render(request, 'homepage.html',{'user':user,'user1':user1,'success':success})
```

2.Details.py

```
def details(request,name):
  success=False
  if request.method=='GET':
      hcom=usercom.objects.all().filter(hname=name)
      my_data=dinner.objects.all()
      for d in my_data:
          if d.hname==name:
              user=dinner.objects.get(hname=d.hname)
      return render(request,'details.html',{'user':user,'hcom':hcom})

  if request.method=='POST':
     if 'BOOK' in request.POST:
      hcom = usercom.objects.all().filter(hname=name)
      hname=request.POST.get('hname')
      user = dinner.objects.get(hname=hname)
      obj=book()
      obj.hname=request.POST.get('hname')
      obj.phone=request.POST.get('phone')
      obj.date=request.POST.get('date')
      obj.person=request.POST.get('person')
      obj.save()
      success=True
      return render(request, 'details.html', {'success': success,'user':user,'hcom':hcom})
     if 'ADD' in request.POST:

         hname = request.POST.get('hname')
         user = dinner.objects.get(hname=hname)
         obj = usercom()
         obj.hname = request.POST.get('hname')
         obj.pname= request.POST.get('pname')
         obj.comments=request.POST.get('comments')

         obj.save()
         success = True
         hcom = usercom.objects.all().filter(hname=name)
         return render(request, 'details.html', {'success1': success, 'user': user,

         'hcom':hcom})
```

```
def detailsd(request, name):
        success = False
        if request.method == 'GET':
            my_data = dinner.objects.all()
            for d in my_data:
                if d.hname == name:
                    user = dinner.objects.get(hname=d.hname)
            return render(request, 'detailsd.html', {'user': user})
        if request.method == 'POST':
            hname = request.POST.get('hname')
            user = dinner.objects.get(hname=hname)
            obj = book()
            obj.hname = request.POST.get('hname')
            obj.phone = request.POST.get('phone')
            obj.date = request.POST.get('date')
            obj.person = request.POST.get('person')
            obj.save()
            success = True
            return render(request, 'detailsd.html', {'success': success, 'user': user})
```

3.Dates.py

```
def dates(request):

    if request.method == 'GET':

        my_data = dinner.objects.all()
        ni="northindian"
        return render(request, "dates.html", {'my_data': my_data,'ni':ni})
    if request.method == 'POST':
        veg = request.POST.get('example1')
        nonveg = request.POST.get('example2')
        drinks = request.POST.get('example3')
        if veg == 'on' and nonveg == 'on' and drinks == 'on':
            name = "NON VEG & DRINKS & DRINKS"
            user = dinner.objects.all().filter(typef='both').filter(drinks='yes')
            return render(request, 'datesc.html', {'my_data': user, 'name': name})
        elif veg == 'on' and nonveg == 'on':
            name = "NON VEG & VEG"
            user = dinner.objects.all().filter(typef='both')
            return render(request, 'datesc.html', {'my_data': user, 'name': name})
        elif veg == 'on' and drinks == 'on':
            name = "VEG & DRINKS"
            user = dinner.objects.all().filter(typef='veg').filter(drinks='yes')
            user1 = dinner.objects.all().filter(typef='both').filter(drinks='yes')
            return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

             'name': name})
        elif nonveg == 'on' and drinks == 'on':
            name = "NON VEG & DRINKS"
            user = dinner.objects.all().filter(typef='non-veg').filter(drinks='yes')
            user1 = dinner.objects.all().filter(typef='both').filter(drinks='yes')
            return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

              'name': name})
        elif veg == 'on':
            name = "VEG"
```

```
            user = dinner.objects.all().filter(typef='veg')
            user1 = dinner.objects.all().filter(typef='both')
            return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

             'name': name})
        elif nonveg == 'on':
            name = "NON VEG"
            user = dinner.objects.all().filter(typef='non-veg')
            user1 = dinner.objects.all().filter(typef='both')
            return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

             'name': name})
        elif drinks == 'on':
            name = "Drinks"
            user = dinner.objects.all().filter(drinks='yes')
            return render(request, 'datesc.html', {'my_data': user, 'name': name})


def datesc(request, name):
        success = False
        if request.method == 'GET':


            user = dinner.objects.all().filter(cusine1=name)
            user2 = dinner.objects.all().filter(cusine2=name)
            user3 = dinner.objects.all().filter(cusine3=name)
            user4 = dinner.objects.all().filter(cusine4=name)
            return render(request, 'datesc.html', {'name1':name,'my_data':
user,'my_data2':user2,'my_data3':user3,'my_data4':user4})
        if request.method == 'POST':
            veg = request.POST.get('example1')
            nonveg = request.POST.get('example2')
            drinks = request.POST.get('example3')
            if veg == 'on' and nonveg == 'on' and drinks == 'on':
                name = "NON VEG & DRINKS & DRINKS"
                user = dinner.objects.all().filter(typef='both').filter(drinks='yes')
                return render(request, 'datesc.html', {'my_data': user, 'name': name})
            elif veg == 'on' and nonveg == 'on':
                name = "NON VEG & VEG"
                user = dinner.objects.all().filter(typef='both')
                return render(request, 'datesc.html', {'my_data': user, 'name': name})
            elif veg == 'on' and drinks == 'on':
                name = "VEG & DRINKS"
                user = dinner.objects.all().filter(typef='veg').filter(drinks='yes')
                user1 = dinner.objects.all().filter(typef='both').filter(drinks='yes')
                return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

              'name': name})
            elif nonveg == 'on' and drinks == 'on':
                name = "NON VEG & DRINKS"
                user = dinner.objects.all().filter(typef='non-veg').filter(drinks='yes')
                user1 = dinner.objects.all().filter(typef='both').filter(drinks='yes')
                return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

             'name': name})
            elif veg == 'on':
                name = "VEG"
                user = dinner.objects.all().filter(typef='veg')
                user1 = dinner.objects.all().filter(typef='both')
                return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,
```

```
            'name': name})
        elif nonveg == 'on':
            name = "NON VEG"
            user = dinner.objects.all().filter(typef='non-veg')
            user1 = dinner.objects.all().filter(typef='both')
            return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

             'name': name})
        elif drinks == 'on':
            name = "Drinks"
            user = dinner.objects.all().filter(drinks='yes')
            return render(request, 'datesc.html', {'my_data': user, 'name': name})
```

4.Familydinner.py

```
def dinnerreg(request):
    global saved1
    obj=dinner()
    if request.method == 'GET':
        return render(request, 'dinnerreg.html')
    else:
        obj.hname = request.POST.get('hname')

        if 'h_pic_1' in request.FILES:
            h_pic_1 = request.FILES['h_pic_1']
        else:
            h_pic_1 = None
        obj.himage1=h_pic_1
        if 'h_pic_2' in request.FILES:
            h_pic_2 = request.FILES['h_pic_2']
        else:
            h_pic_2 = None
        obj.himage2=h_pic_2
        obj.hadd = request.POST.get('hadd')
        obj.hpp=request.POST.get('hpp')
        obj.cusine1=request.POST.get('cusine1')
        obj.cusine2 = request.POST.get('cusine2')
        obj.cusine3 = request.POST.get('cusine3')
        obj.cusine4 = request.POST.get('cusine4')
        obj.typef=request.POST.get('typef')
        obj.drinks=request.POST.get('drinks')
        obj.save()
        saved1 = True
        return render(request, 'dinnerreg.html', {'success': saved1})



def familydinnerc(request, name):
    success = False
    if request.method == 'GET':
        user = dinner.objects.all().filter(cusine1=name)
        user2 = dinner.objects.all().filter(cusine2=name)
        user3 = dinner.objects.all().filter(cusine3=name)
        user4 = dinner.objects.all().filter(cusine4=name)
        return render(request, 'familydinnerc.html',
                      {'name1': name, 'my_data': user, 'my_data2': user2, 'my_data3': user3,
'my_data4': user4})
    if request.method == 'POST':
        veg = request.POST.get('example1')
```

```
        nonveg = request.POST.get('example2')
        drinks = request.POST.get('example3')
        if veg == 'on' and nonveg == 'on' and drinks == 'on':
            name = "NON VEG & DRINKS & DRINKS"
            user = dinner.objects.all().filter(typef='both').filter(drinks='yes')
            return render(request, 'datesc.html', {'my_data': user, 'name': name})
        elif veg == 'on' and nonveg == 'on':
            name = "NON VEG & VEG"
            user = dinner.objects.all().filter(typef='both')
            return render(request, 'datesc.html', {'my_data': user, 'name': name})
        elif veg == 'on' and drinks == 'on':
            name = "VEG & DRINKS"
            user = dinner.objects.all().filter(typef='veg').filter(drinks='yes')
            user1 = dinner.objects.all().filter(typef='both').filter(drinks='yes')
            return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

 'name': name})
        elif nonveg == 'on' and drinks == 'on':
            name = "NON VEG & DRINKS"
            user = dinner.objects.all().filter(typef='non-veg').filter(drinks='yes')
            user1 = dinner.objects.all().filter(typef='both').filter(drinks='yes')
            return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

'name': name})
        elif veg == 'on':
            name = "VEG"
            user = dinner.objects.all().filter(typef='veg')
            user1 = dinner.objects.all().filter(typef='both')
            return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

 'name': name})
        elif nonveg == 'on':
            name = "NON VEG"
            user = dinner.objects.all().filter(typef='non-veg')
            user1 = dinner.objects.all().filter(typef='both')
            return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

 'name': name})
        elif drinks == 'on':
            name = "Drinks"
            user = dinner.objects.all().filter(drinks='yes')
            return render(request, 'datesc.html', {'my_data': user, 'name': name})


def familydinner(request):
    if request.method == 'GET':
        my_data = dinner.objects.all()
        return render(request, "familydinner.html", {'my_data': my_data})
    if request.method == 'POST':
        veg = request.POST.get('example1')
        nonveg = request.POST.get('example2')
        drinks = request.POST.get('example3')
        if veg == 'on' and nonveg == 'on' and drinks == 'on':
            name = "NON VEG & DRINKS & DRINKS"
            user = dinner.objects.all().filter(typef='both').filter(drinks='yes')
            return render(request, 'datesc.html', {'my_data': user, 'name': name})
        elif veg == 'on' and nonveg == 'on':
            name = "NON VEG & VEG"
            user = dinner.objects.all().filter(typef='both')
            return render(request, 'datesc.html', {'my_data': user, 'name': name})
        elif veg == 'on' and drinks == 'on':
            name = "VEG & DRINKS"
```

```
            user = dinner.objects.all().filter(typef='veg').filter(drinks='yes')
            user1 = dinner.objects.all().filter(typef='both').filter(drinks='yes')
            return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

 'name': name})
        elif nonveg == 'on' and drinks == 'on':
            name = "NON VEG & DRINKS"
            user = dinner.objects.all().filter(typef='non-veg').filter(drinks='yes')
            user1 = dinner.objects.all().filter(typef='both').filter(drinks='yes')
            return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

 'name': name})
        elif veg == 'on':
            name = "VEG"
            user = dinner.objects.all().filter(typef='veg')
            user1 = dinner.objects.all().filter(typef='both')
            return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

'name': name})
        elif nonveg == 'on':
            name = "NON VEG"
            user = dinner.objects.all().filter(typef='non-veg')
            user1 = dinner.objects.all().filter(typef='both')
            return render(request, 'datesc.html', {'my_data': user, 'my_data2': user1,

 'name': name})
        elif drinks == 'on':
            name = "Drinks"
            user = dinner.objects.all().filter(drinks='yes')
            return render(request, 'datesc.html', {'my_data': user, 'name': name})
```

5.wedding.py

```
def wedding(request):
    return render(request,'wedding.html')
```

6.Aboutus.py

```
def contactus(request):
    today=date.today()

    return render(request,'contactus.html',{'today':today})
saved1=False


def aboutus(request):
    return render(request,'aboutus.html')
```

7 decor.py

```
def decor(request):
    global saved
    DecorModel=decorModel()

    if request.method=='GET':
        return render(request,"decor.html")
```

```
    else:
        decorname=request.POST.get('decorname')

        if 'profile_pic_1' in request.FILES:
            profile_pic_1 = request.FILES['profile_pic_1']
        else:
            profile_pic_1=None

        print(decorname,profile_pic_1)

        DecorModel.decorname=decorname
        DecorModel.image1=profile_pic_1
        DecorModel.save()

        saved=True

        return render(request,'decor.html',{'success':saved})
```

8. venue.py

```
def venue(request):
    global saved
    VenueModel=venueModel()

    if request.method=='GET':
        return render(request,"venue.html")
    else:
        venuename=request.POST.get('venuename')

        if 'profile_pic_1' in request.FILES:
            profile_pic_1 = request.FILES['profile_pic_1']
        else:
            profile_pic_1=None
        if 'profile_pic_2' in request.FILES:
            profile_pic_2 = request.FILES['profile_pic_2']
        else:
            profile_pic_2=None
        if 'profile_pic_3' in request.FILES:
            profile_pic_3 = request.FILES['profile_pic_3']
        else:
            profile_pic_3=None
        if 'profile_pic_4' in request.FILES:
            profile_pic_4 = request.FILES['profile_pic_4']
        else:
            profile_pic_4=None
        if 'profile_pic_5' in request.FILES:
            profile_pic_5 = request.FILES['profile_pic_5']
        else:
            profile_pic_5=None
        if 'profile_pic_6' in request.FILES:
            profile_pic_6 = request.FILES['profile_pic_6']
```

```
        else:
            profile_pic_6=None

        print(venuename,profile_pic_1)

        VenueModel.venuename=venuename
        VenueModel.image1=profile_pic_1
        VenueModel.image2 = profile_pic_2
        VenueModel.image3 = profile_pic_3
        VenueModel.image4 = profile_pic_4
        VenueModel.image5 = profile_pic_5
        VenueModel.image6 = profile_pic_6
        VenueModel.city=request.POST.get('city')
        VenueModel.save()

        saved=True

        return render(request,'venue.html',{'success':saved})
def portfolio(request,name):
    v_data=venueModel.objects.all().filter(venuename=name)
    return render(request,'portfolio.html',{'my_data':v_data})
def portfoliow(request,name):
    v_data=venueModel.objects.all().filter(venuename=name)
    return render(request,'portfoliow.html',{'my_data':v_data})
```

9.party.py

```
def partys(request, name):
    success = False
    if request.method == 'GET':
        user = venueModel.objects.all().filter(city=name)
        return render(request, 'partys.html',
                      {'name1': name, 'my_data': user})

def party(request):
    if request.method=='GET':

        v_data=venueModel.objects.all()
        return render(request,"party.html",{'my_data':v_data})

def partyd(request,name):
    if request.method=='GET':
        d_data=decorModel.objects.all()
        return render(request, "partyd.html", {'my_data': d_data,'vname':name})
def partyreg1(request,dname,vname):
    if request.method=='GET':
        today=date.today()
        return render(request, "partyreg1.html", {'dname':dname , 'vname':vname,'today':today})
    success=False
    if request.method=='POST':
        obj = partyreg()
        pdate=request.POST.get('date')
        hname=request.POST.get('vname')
        test=partyreg.objects.all().filter(pdate=pdate).filter(venuename=hname)
        var="no"
        for d in test:
```

```
            var="yes"
        if var=="yes":
            v_data = venueModel.objects.all()
            success=True
            return render(request, 'party.html', {'success1': success, 'my_data': v_data})
        else:
            v_data = venueModel.objects.all()

            obj.ptype = "party"
            obj.pname = request.POST.get('name')
            obj.pemail = request.POST.get('email')
            obj.pdate=request.POST.get('date')
            obj.phone = request.POST.get('mobile')
            obj.pguest=request.POST.get('guest')
            obj.venuename=request.POST.get('vname')
            obj.decorname=request.POST.get('dname')
            obj.ptype="party"
            phh=request.POST.get('photo')
            dss=request.POST.get('dj')

            if phh == 'on':
                obj.photo = request.POST.get('photo')
            else:
               obj.photo = "none"
            if dss == 'on':
                obj.ds = request.POST.get('dj')
            else:
                obj.ds = "none"
            obj.save()
            success=True

            return render(request,'party.html',{'success':success,'my_data':v_data})
```

10. wedding.py

```
def weddings(request, name):
    success = False
    if request.method == 'GET':
        user = venueModel.objects.all().filter(city=name)
        return render(request, 'weddings.html',
                      {'name1': name, 'my_data': user})



def wedding(request):
    if request.method=='GET':

        v_data=venueModel.objects.all()
        return render(request,"wedding.html",{'my_data':v_data})

def weddingd(request,name):
    if request.method=='GET':
        d_data=decorModel.objects.all()
        return render(request, "weddingd.html", {'my_data': d_data,'vname':name})
def weddingreg(request,dname,vname):
    if request.method=='GET':

        return render(request, "weddingreg.html", {'dname':dname , 'vname':vname})
```

```
    success=False
    if request.method=='POST':
        obj=partyreg()
        pdate = request.POST.get('date')
        test = partyreg.objects.all().filter(pdate=pdate)
        var = "no"
        for d in test:
            var = "yes"
        if var == "yes":
            v_data = venueModel.objects.all()
            success = True
            return render(request, 'party.html', {'success1': success, 'my_data': v_data})
        else:
         obj.ptype="wedding"
         obj.pname=request.POST.get('name')
         obj.pemail=request.POST.get('email')
         dss=request.POST.get('dj')
         obj.phone=request.POST.get('mobile')
         obj.pdate=request.POST.get('date')
         obj.pguest=request.POST.get('guest')
         obj.venuename=request.POST.get('vname')
         obj.ptype = "wedding"
         obj.decorname=request.POST.get('dname')
         phh=request.POST.get('photo')
         if phh=='on':
          obj.photo=request.POST.get('photo')
         else:
            obj.photo="none"
         if dss=='on':
          obj.ds=request.POST.get('dj')
         else:
          obj.ds="none"
         obj.save()
         success=True
         v_data = venueModel.objects.all()
         return render(request,'wedding.html',{'success':success,'my_data':v_data})
```

11.Gallery.py

```
def gallery(request):
    user=gallerym.objects.all()
    return render(request,'gallery.html',{'user':user})
def galleryreg(request):
    if request.method=='GET':
      return render(request,'galleryreg.html')
    success=False
    if request.method=='POST':
        obj=gallerym()
        obj.group=request.POST.get('group')
        obj.work=request.POST.get('work')
        if 'profile_pic_1' in request.FILES:
            profile_pic_1 = request.FILES['profile_pic_1']
        else:
            profile_pic_1 = None
        obj.image=profile_pic_1
        obj.save()
```

```
        success=True
        return render(request,'galleryreg.html',{'success':success})
```