



SE 505 : Software Project Lab - 2

Save The Date : A Party Planning Application

(SPL-2 Final Report, 2022)

Submitted By :

1) Mushfiqur Rahman Chowdhury

Roll : 1110

Session : 2018-19

2) Shafiq-us Saleheen

Roll : 1125

Session : 2018-19

Supervised By :

Dr. Sumon Ahmed

Associate Professor

Institute of Information Technology,

University of Dhaka

SIGNATURE PAGE

Project : Save The Date : A Party Planning Application

Author : Mushfiqur Rahman Chowdhury
(Roll - 1110)

Shafiq-us Saleheen
(Roll - 1125)

Date Submitted : June 15th, 2022

Supervised By : Dr. Sumon Ahmed
Associate Professor
Institute of Information Technology,
University of Dhaka.

Supervisor's Approval :

(Signature of Dr. Sumon Ahmed)

LETTER OF TRANSMITTAL

BSSE 3rd SPL Committee,
Institute of Information Technology,
University of Dhaka

Subject: Submission of the final report on "Save the date".

Dear Sir,

With due respect, we're pleased to submit herewith our report on "Save the date: A party planning application". In writing this report, we've followed the instructions you've given us in the class. We've applied the relevant topics that we learned throughout the course. Though the report may contain some errors, know that we've tried our best to make an approvable one. We would be grateful if you overlooked our mistakes and accepted our efforts.

Sincerely yours,

Mushfiqur Rahman Chowdhury &
Shafiq-us Saleheen
BSSE 11th Batch,
Institute of Information Technology,
University of Dhaka.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our project supervisor, Dr. Sumon Ahmed Sir, Associate Professor, Institute of Information Technology, University of Dhaka, for giving us an opportunity to go on with this project and providing unconditional support throughout the semester. His invaluable guidance and motivation helped us make this project a successful one. It was an honor to work under his supervision.

SOFTWARE AVAILABILITY

[saleheenshafi9/SaveTheDate: SaveTheDate is a party planning application \(github.com\)](#)

TABLE OF CONTENTS

CHAPTER 1	9
INTRODUCTION OF Save the date	9
1.1 Overview	9
1.2 Objectives	9
1.3 Motivation	10
CHAPTER 2	11
BACKGROUND OF Save the date	11
2.1 Three-tier Architecture	11
2.1.1 Presentation Tier : React JS Front-end Framework	11
2.1.2 Application Tier : Django Back-end Framework	12
2.1.3 Data Tier : MySQL Database	13
2.2 REST API	13
CHAPTER 3	14
DESCRIPTION OF Save the date	14
3.1 Login & Signup System	14
3.2 Profile Management System	15
3.2.1 Service Provider	15
3.2.2 Customer	16
3.3 Party Booking	17
3.4 Appointment Scheduling	19
3.5 Recommendation System	20
CHAPTER 4	22
IMPLEMENTATION & TESTING	22
4.1 Front-End Implementation	22
4.1.1 Styling	22
4.1.2 Components Structure	23
4.1.3 Profile Switching	25
4.1.4 Handling Cart Items	27
4.1.5 Testing	28
4.2 Back-End Implementation	29

4.2.1 Sign Up & Login	29
4.2.2 Recommendation System	30
4.2.3 Party Booking API	32
4.2.4 Profile Update	34
CHAPTER 5	36
USER MANUAL of Save the date	36
5.1 Authentication & Authorization	36
5.2 Navigation Bar	37
5.3 Customer View	38
5.3.1 Booking Section	38
5.3.2 Food Item Selection	39
5.3.3 Recommendation System	40
5.4 Provider View	41
SaveTheDate- Demo Video for User Manual	42
CHAPTER 6	43
CHALLENGES FACED in implementing	43
6.1 Learning to Collaborate & Distribute	43
6.2 Requirements Engineering & Data Collection	43
6.3 Version Updates	44
6.4 CRUD Operations	44
6.5 Back-end Complications	45
6.6 Recommendation Parameter Settings	45
CHAPTER 7	46
FUTURE WORK & CONCLUSION of Save the date	46
7.1 Future Work	46
7.2 Conclusion	47
REFERENCES	48

LIST OF TABLES

Component Functionalities	23
URL Distribution of Party	32
URL Distribution of Profile	35
Navigation Routings	36

LIST OF FIGURES

Django File Systems	12
RESTful API Mechanism	13
Login & Signup Page	14
Whitehall Convention Hall Edit Venue Profile	15
Whitehall Convention Hall Venue Profile	16
Customer Profile	17
Cart System & Payment Simulation	18
Party Booking Activity Diagram	18
Appointment Scheduling Flow	19
UI to Get Recommendation Parameters	20
Generated Plan for the Customer	21
Dependencies from package.json	22
UI of User type Form	26
Implementation of Private Routing	26
Implementation of Sorting Items into Context	27
Implementation of Party Booking (CRUD Operations)	28
Sign up Completion using REST	29

Get Access & Refresh Token	30
Recommendation Models	31
Party API Models	33
Profile API Models	34
HomePage of Save the date	36
Sign in to Login/Register Page	37
Customer Dashboard	38
Payment Section Manual	39
Food Selection Manual	39
Diagram of Recommendation Manual	40
Generated Plan for the Customer	40
Provider Profile Page Manual	41
Calendar	41
Edit Profile Page	42
Syntax Change of react-router-dom	44

CHAPTER 1

INTRODUCTION OF Save the date

1.1 Overview

SaveTheDate is a party planning web application that was developed to connect different party related service providers to their customers. A lot of birthday and wedding parties take place around us and different kinds of party related services such as venue, catering, photographers and decorators are needed by the customers. The searching and booking procedure of party related services in the traditional way often become more complicated from both the customer and service provider view-point. To make things easier, we introduced our web based application the 'SaveTheDate' where customers can book services, view service details, set appointments with the service providers they are interested in to further discuss the event. We are hopeful that our application will make the process of organizing parties easier and also bring in more customers for service providers.

1.2 Objectives

- Our application will provide all the party related service providers, all in one place such as- venues, catering services, decorators, multimedia content makers etc.
- It will help generate the full planning of the event automatically.
- It will provide service recommendations according to the user information and help make it easy for the host to search for services.
- It will help customers to keep track of the event planning and other customizations.

1.3 Motivation

Nowadays, people often are busy with their work, business etc. On top of that, if someone also needs to organize an event for any occasion, it adds extra pressure to it. Even though one decides to take up the challenge, it has become a major issue to keep track of the budget and manipulate it all the time to see it mustn't exceed. These hassles make an occasion more and more timid. To buzz through this issue, we've come up with a plan that may reduce all the difficulties for a host and they could just save the date and move, the rest of it will be handled automatically by our application. Hence, we have 'Save The Date'.

This application has necessary features that will keep the host ahold of all the activities. They can simply enjoy the occasion without unnecessary thinking about the venue, foods, guest accommodations etc. Thus, one can enjoy their own party to the fullest.

CHAPTER 2

BACKGROUND OF Save the date

Our project is a web application that is built on several web technologies. So, we had to do a lot of background research to have a better understanding of the full process. Let's discuss them-

2.1 Three-tier Architecture

Our project 'SaveTheDate' was built on a three-tier architecture. The architecture organizes applications into three logical and physical computing tiers-

- The **Presentation** tier, or user interface;
- The **Application** tier, where data is processed; and
- The **Data** tier, where the data associated with the application is stored and managed.

2.1.1 Presentation Tier : React JS Front-end Framework

For our presentation tier, or the frontend, we are using **ReactJS framework**. It is a front-end framework which is highly efficient in keeping clean project structure and maintaining component-based modeling. It's also easier to learn than other frameworks like Angular or Vue.

To build a more interactive user interface, React provides a high-quality UI concept. It also has technical terms such as, React hooks, reducers etc. to write custom components. React comes with JSX, an optional syntax extension, which makes it possible to write your own components.

React web framework, on the other hand, is currently being utilized by famous companies including Netflix, Paypal, NASA, BBC, Lyft, and New York Times to name just a few. Thus, we used it.

2.1.2 Application Tier : Django Back-end Framework

For our application tier or the backend, we are using **Django Framework**. Django is a high-level Python web framework. It uses REST API to receive requests from the presentation tier and return responses accordingly. It can work with any client-side framework, and can deliver content in almost any format. It has a component-based architecture model to provide higher security and scalability. In a website, a web application waits for HTTP requests from the client. When a request is received the application works out what is needed based on the URL and possibly information in POST data or GET data. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request.

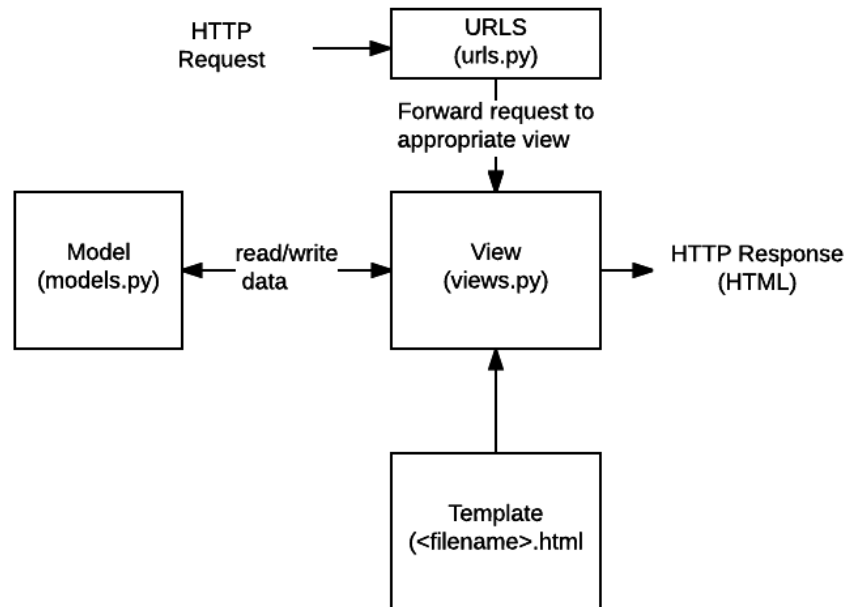


FIGURE-1 : Django File System (Mozilla Developer)

2.1.3 Data Tier : MySQL Database

Our database contains structured data and uses **MySQL** which is a Relational Database Management System. MySQL has been used because it provides comprehensive support for every application development need. A Unique storage-engine architecture allows database professionals to configure the MySQL database server specifically for particular applications, with the end result being amazing performance results. Thus, we used MySQL.

2.2 REST API

The Django Framework uses REST API to receive requests from the presentation tier and return responses accordingly. The REST API is an Application Programming Interface that allows interaction of RESTful web services. The full form of REST API is Representational State Transfer Application Programming Interface. It completes CRUD operations through POST, GET, PUT, DELETE etc. methods. It helps connect the front-end with the back-end and thus store data securely.

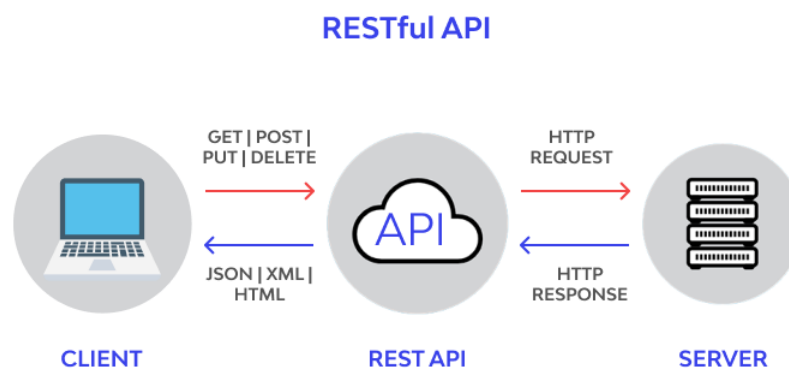


FIGURE-2 : RESTful API Mechanism (Novikov)

CHAPTER 3

DESCRIPTION OF Save the date

The key features of our project include a login and signup system, profile visiting and profile management system, party booking system, browsing facility to look at different services, appointment setting system and a system to show the progress of the currently booked party.

3.1 Login & Signup System

New users of the system who do not have any existing account can get registered in the system by entering user information. We have built a form using our frontend framework which takes as input the following fields: **username, usertype, email, password, FirstName** and **LastName**. The data received using the form is then added to the database by sending a POST request to the database. The user data is added to the database. While logging in, the user provides their username and previously set password to log in. After login, the system automatically routes to the profile page related to the user.

The figure displays two side-by-side screenshots of a web application's authentication interface. The left screenshot shows the 'Login' page, featuring a 'Login' button at the top left and a 'Sign Up' button at the top right. Below these buttons are two input fields: 'Username' and 'Password'. A 'Login' button is positioned below the password field. At the bottom, there is a link 'Don't have an account? Register Here.' and a 'Sign in with Google' button. The right screenshot shows the 'Signup' page, with 'Login' and 'Sign Up' buttons at the top. The form includes fields for 'First Name', 'Last Name', 'Email', 'Username', and 'User Type' (with a 'Customer' dropdown menu). Below these are fields for 'Phone Number', 'Password', and 'Confirm Password'. A 'Register' button is located at the bottom of the form, along with a link 'Already have an account? Login Here.'

FIGURE-3 : Login & Signup Page

3.2 Profile Management System

There will be basically two user types such as-

- Customers
- Providers (Venue, Caterer, Decorator, Contentmaker)

Each type of user can build their own profile where they will be able to update their info and upload photos.

3.2.1 Service Provider

Service providers can create a gallery in their profile where they will upload multiple photos so that the customers can view those images and learn about the provider. A service provider will have a title and description field. The service provider will be able to update the description field which will help customers understand their service.

The screenshot shows a web interface for editing venue information. At the top is a yellow navigation bar with the 'SaveTheDate' logo and links for Home, Services, Blogs, Contact Us, and Sign Out. Below the navigation bar is the 'Edit Venue Information' section. It contains four text input fields: 'Location' (placeholder: 'Where's the venue situated?'), 'Capacity' (placeholder: 'Approx. number of guests?'), 'Title' (placeholder: 'Venue name?'), and 'Description' (placeholder: 'Details of the venue?'). To the right of these fields is a file upload area with a camera icon, a 'Choose File' button, and the text 'No file chosen'. At the bottom right of the form is a green 'Save' button with a download icon. A small 's' is visible at the bottom left of the form area.

FIGURE-4 : Whitehall Convention Hall Edit Venue Profile

Whitehall Convention Hall

Mohammadpur, Dhaka
 1200 People
 80000 BDT

About Venue

Enjoyable and comfortable atmosphere in the heart of Dhanmondi. The White Hall Convention and Buffet is dedicated to providing the highest quality of service in order to create a fine dining atmosphere. We offer a grand buffet experience through our exquisite food in the warm & friendly ambiance complemented by the personalized customer service provided by our exceptional waitstaff. In our all you can eat restaurant, with our food quality and taste, we have achieved a reputation and trust of becoming the best buffet restaurant in Dhanmondi with our ultimate goal towards becoming the most fancied buffet in Dhaka city.

FIGURE-5 : Whitehall Convention Hall Venue Profile

3.2.2 Customer

If userType is selected, based on the selection, they will be redirected to a different kind of profile which has unique dashboard items. Similarly, a customer also has a unique profile to see the items he requires, booking situation, past bookings, upcoming events etc. data on a dashboard of its own.

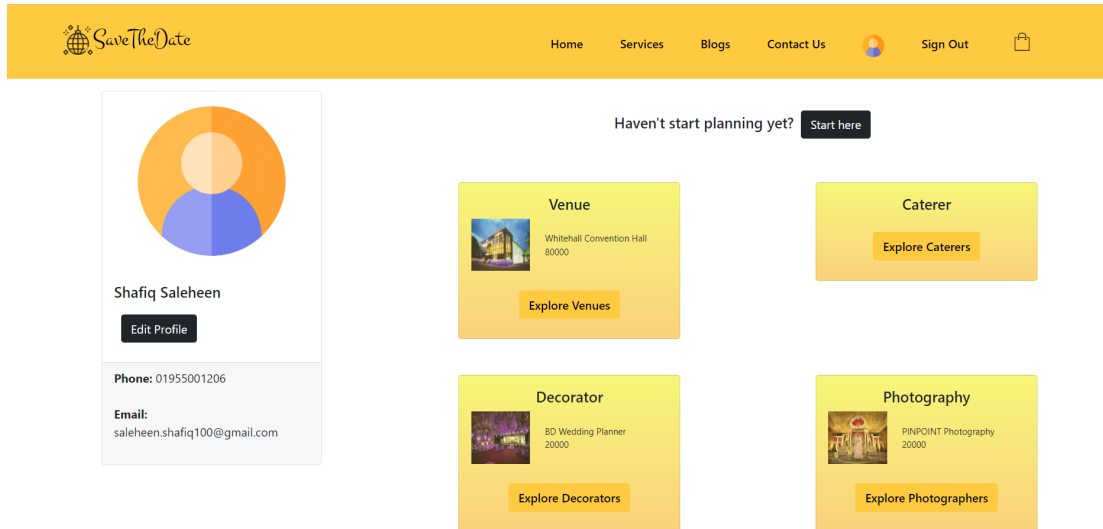


FIGURE-6 : Customer Profile

3.3 Party Booking

If a customer logs into the system, they can explore different vendors provided by our services. They can click on any items and see their details. If preferred, they can add vendors to their cart, to generate the cost of the items. Also the selected items will be automatically added to their user profile, from where they can easily proceed to the checkout section.

From there, they can easily go to the checkout page, which will have detailed info of payment and other details. Upon completion of the payment process, they will receive a message containing 'Successful' information.

Once they complete the checkout and payment, the vendors will be automatically added to the profile of the customer for further usage.

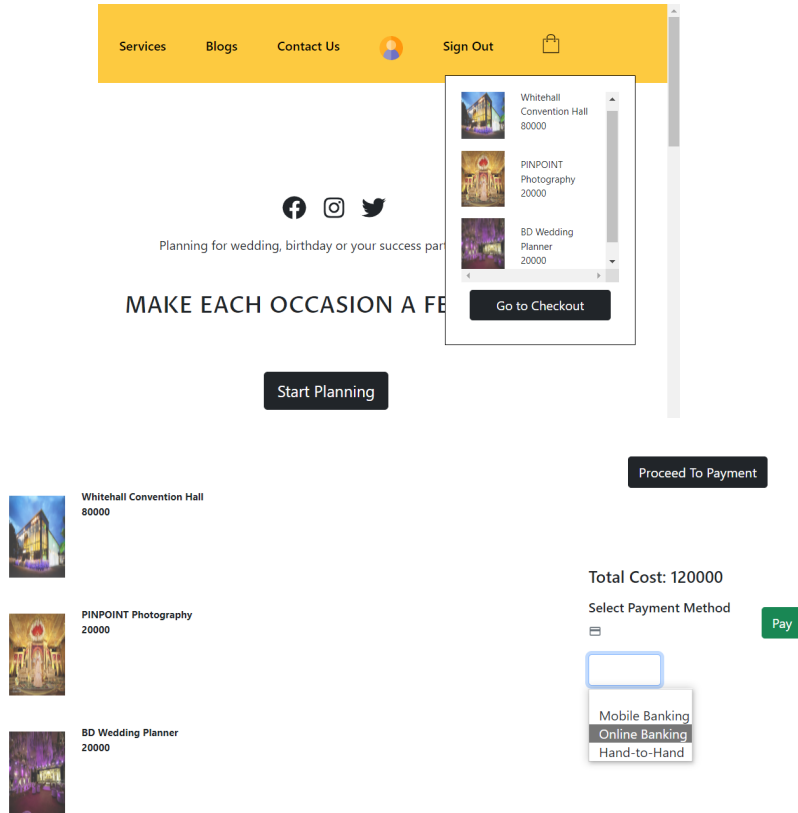


FIGURE-7 : Cart System & Payment Simulation

Here's a complete flowchart of the booking system including payment-

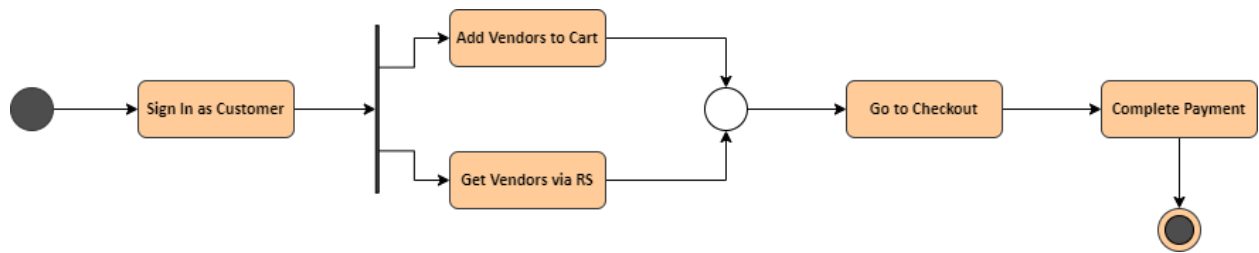


FIGURE-8 : Party Booking Activity Diagram

3.4 Appointment Scheduling

A customer can request an appointment with any service provider. The customer can request any two types of appointments such as, phone call and visit. If the service provider accepts the requested appointment, then an appointment will be scheduled where both customers and service providers will be able to discuss the details of the service.

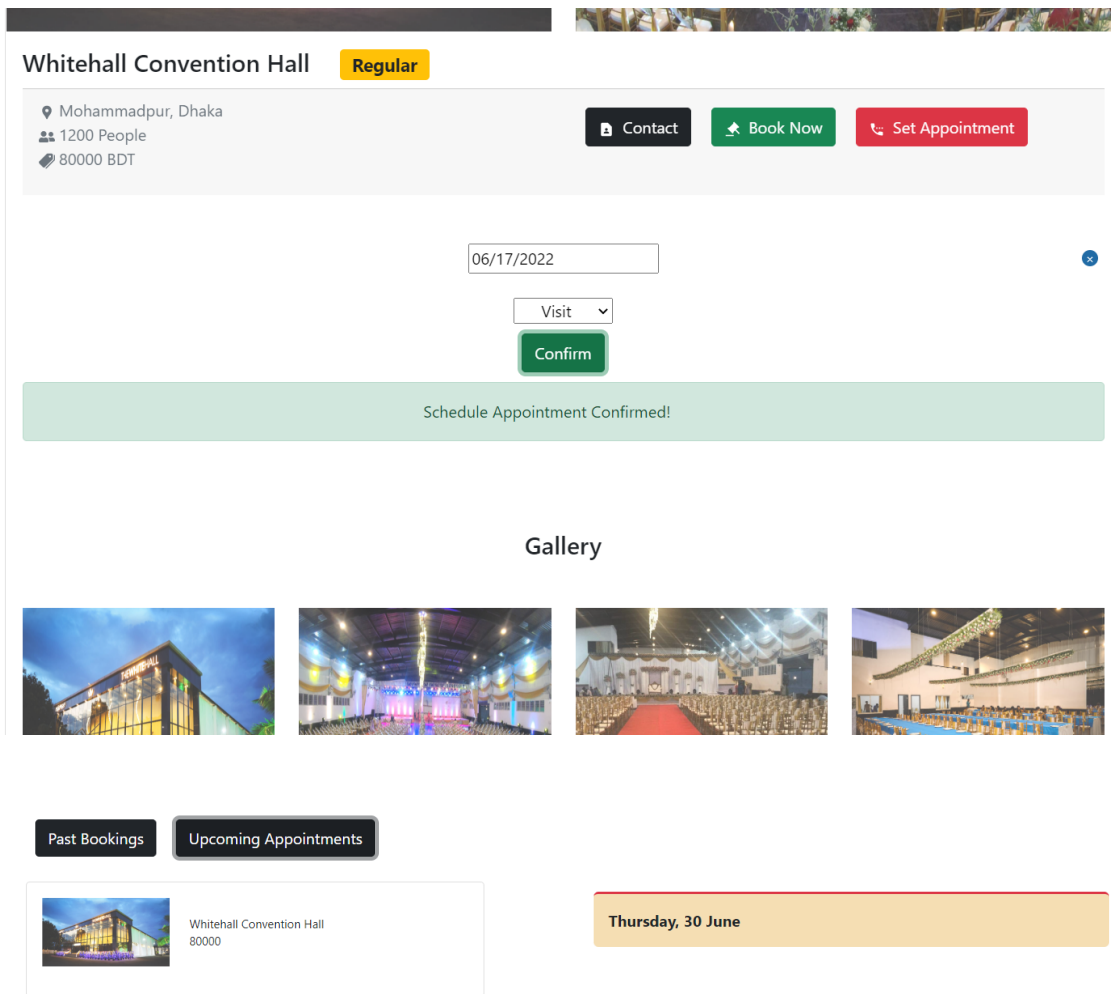


FIGURE-9 : Appointment Scheduling Flow

The appointments made by the customer will be added to both the customer and the provider's profile. Whenever one signs in, it'll automatically popup as a reminder.

3.5 Recommendation System

Our most exciting feature is the recommendation system where the user provides certain parameters as input such as-

1. Date,
2. Number of guests,
3. Budget and
4. City location.

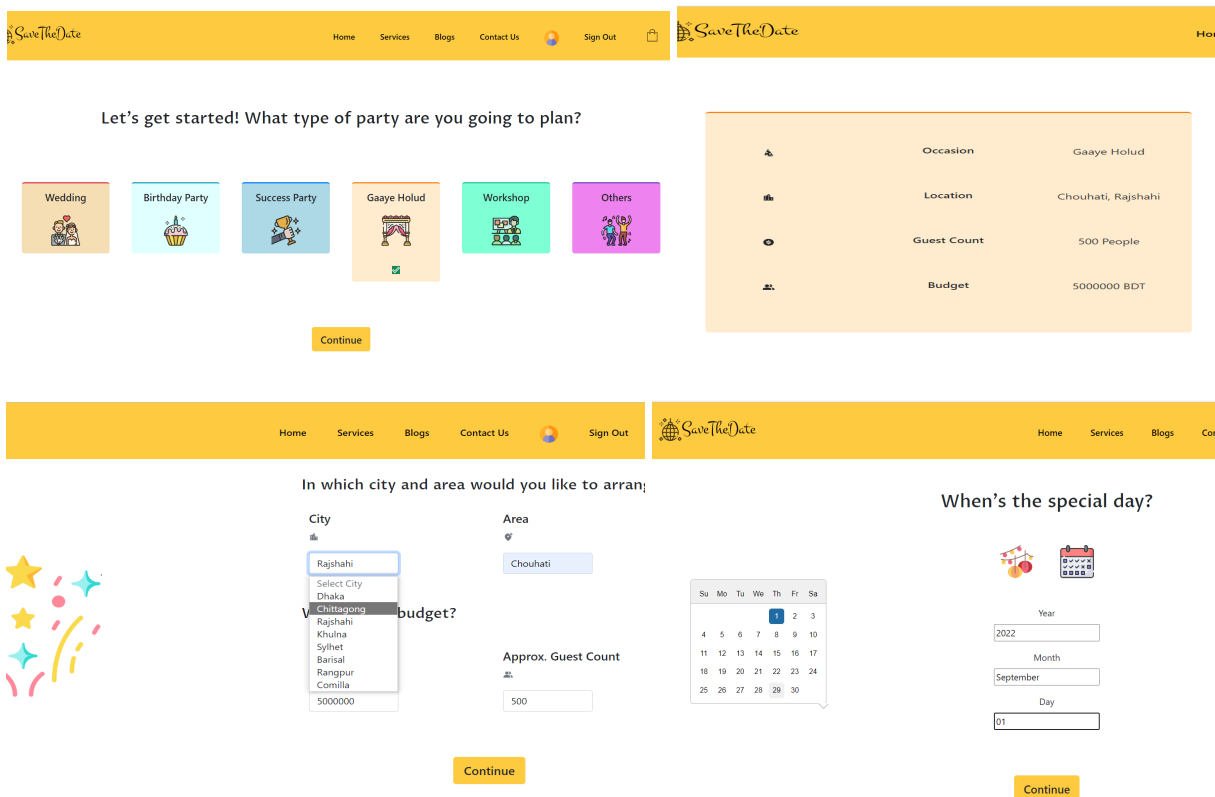


FIGURE-10 : UI to Get Recommendation Parameters

The recommendation system uses the **K-Nearest Neighbor** algorithm to recommend different services to the user based on the input parameters they provide. K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique. It stores all available data and classifies a new datapoint based on similarity. The system recommends different services to the user based on the past bookings of parties.

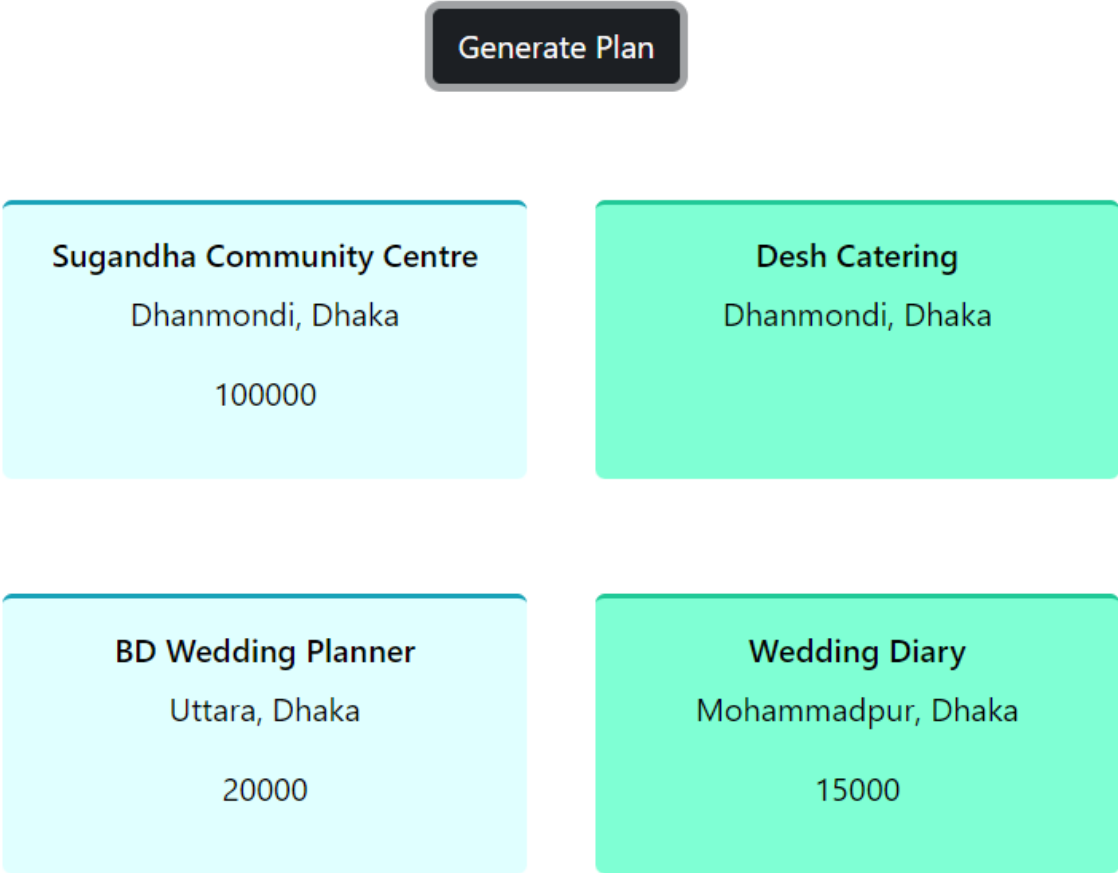


FIGURE-11 : Generated Plan for the Customer

The customer can choose the recommendation simply by clicking on the cards and going to further details. This system will help the customer to reduce the hassle of searching. It will also reduce time waste for them.

CHAPTER 4

IMPLEMENTATION & TESTING

4.1 Front-End Implementation

We've used a JavaScript based framework, React JS for front-end development. React is a very popular framework and has component-based modeling to implement user interface with more ease.

4.1.1 Styling

For styling, we've incorporated bootstrap 5 library with the application. We've also used different libraries of react to make our application look more attractive. For example-

```
11  "dependencies": {
12    "@fortawesome/fontawesome-svg-core": "^6.1.1",
13    "@fortawesome/free-regular-svg-icons": "^6.1.1",
14    "@fortawesome/free-solid-svg-icons": "^6.1.1",
15    "@fortawesome/react-fontawesome": "^0.1.18",
16    "@reduxjs/toolkit": "^1.8.1",
17    "@syncfusion/ej2-react-schedule": "^20.1.57",
18    "@testing-library/jest-dom": "^5.16.4",
19    "@testing-library/react": "^12.1.4",
20    "@testing-library/user-event": "^13.5.0",
21    "@vitejs/plugin-react-refresh": "^1.3.1",
22    "axios": "^0.26.1",
23    "bootstrap": "^5.1.3",
24    "formik": "^2.2.9",
25    "jwt-decode": "^3.1.2",
26    "react": "^18.0.0",
27    "react-calendar": "^3.7.0",
28    "react-datepicker": "^4.8.0",
29    "react-dom": "^18.0.0",
30    "react-icons": "^4.3.1",
31    "react-phone-number-input": "^3.1.50",
32    "react-redux": "^8.0.0",
33    "react-router": "^6.3.0",
34    "react-router-dom": "^6.3.0",
35    "react-toastify": "^9.0.2",
36    "reactstrap": "^9.0.1",
```

FIGURE-12: Dependencies from package.json

We used react-icons library to fetch icons from different icons providing services such as, font-awesome, material io etc. We also used react-calendar and

react-datepicker to show a more organized version of schedule calendars. Other than date, react-toastify has been used to show more decorated alert messages. We used different components of reactstrap as well for easy readiness of the user interface. We've added fonts from google fonts library.

4.1.2 Components Structure

Our application has following components implementing various features-

Component Name	Functions
Header	It contains the navigation bar which sticks at the top. It is probably the very basis of routings.
Services	It contains all the vendor cards to showcase.
VenueService	It contains various information regarding the venue provider and contains other components like gallery, calendar, editProfiles etc.
CatererService	It contains various information regarding the catering provider and contains other components like gallery, calendar, editProfiles etc.
DecoratorService	It contains various information regarding the decoration provider and contains other components like gallery, calendar, editProfiles etc.
PhotographyService	It contains various information regarding the content maker provider and contains other components like gallery, calendar, editProfiles etc.
Alert	It creates customized alert messages which are used in case of successful or failed task development.
Calendar	It shows a full calendar with info on it.
UpcomingAppointment	It inherits all the appointment confirmations and

	shows it in user profiles.
AppointmentSchedule	It schedules appointments upon customer and provider confirmation.
UserContext	It helps with authentication and routing, using react hooks, such as, useContext(), useState(), useEffect() etc.
CartContext	It helps with adding to cart and flowing of information, using react hooks, such as, useContext(), useState(), useEffect() etc.
CartItems	It has all the cart containers.
CartDropdowns	It creates the shopping icon which will have selected items in it.
CustomerProfile	It has the full customer dashboard UI.
ProviderProfile	It has the full provider dashboard UI.
Helper	It has files such as, getReq(), putReq(), postReq() etc. to help with REST API to connect front-end and back-end.
UseFetch	The function which is called whenever any CRUD operation needs to be operated.
Contact	It contains the feedback form.
EditProfile	It gives access to providers to edit info, basically gives a PUT request to the back-end.
Login	It has the whole login UI and the authentication and validation.
Register	It has the whole registration form UI and the authentication, validation and gives authorization.
Logout	It helps with clearing the access token from the header.

Payment	It helps to complete payment transactions and complete booking, as well as store booking data.
Gallery	It creates the modal view of photos, to give a better view to the customers.
PrivateRoute	It understands profile userType and routes them accordingly.
Map	It contains location info and has longitude and latitude to modify or get whenever a request is made.
Recommendation	It contains the recommendation UI. Know that, the implementation of recommendation algorithms have been done in the back-end. So here only are the containers to show results.

TABLE-1 : Component Functionalities

4.1.3 Profile Switching

Our application has two types of users as we already have mentioned them before. But the interesting part is, we also have five types of provider users as well. So, the routing implementation for this part is a bit complex.

To solve this problem, we had an attribute named User Type in our registration form. So, whenever someone signs up, they'll have to provide a string which matches exactly with the backend implementation of authorization. Based on this string, we've a private routing system which will redirect the user according to their user type to their definite profile. Profiles have been prepared and also routes have been added accordingly. So, let's check these screenshots and code snippets for a better understanding-

We'll never share your email with anyone else.

Username

User Type

- Customer
- Customer
- Venue Provider
- Caterer
- Photographer
- Decorator

Phone Number

Password

Confirm Password

Register

FIGURE-13 : UI of User Type Form

```

    ], [token, navigate, loading])

const NavigateToProfile={()=>{
  currentUser?.userType=="customer" && navigate('../customerProfile')
  currentUser?.userType=="venue" && navigate('../venueProfile')
  currentUser?.userType=="catering" && navigate('../catererProfile')
  currentUser?.userType=="contentmaker" && navigate('../photographyProfile')
  currentUser?.userType=="decorator" && navigate('../decoratorProfile')
}

useEffect(()=>{
  currentUser && setLoggedIn(true)
  !currentUser && setLoggedIn(false)
}, [currentUser])

useEffect(()=>{
  !loggedIn && token && updateToken()
}, [loggedIn])

```

```

/* Private Profiles */
<route path="/customerProfile"
  element={
    <PrivateRoute>
      <CustomerProfile />
    </PrivateRoute>
  } />
<route path="/catererProfile"
  element={
    <PrivateRoute>
      <CatererProfile />
    </PrivateRoute>
  } />
<route path="/venueProfile"
  element={
    <PrivateRoute>
      <VenueProfile />
    </PrivateRoute>
  } />
<route path="/decoratorProfile"
  element={
    <PrivateRoute>
      <DecoratorProfile />
    </PrivateRoute>
  } />
<route path="/photographyProfile"
  element={
    <PrivateRoute>
      <PhotographyProfile />
    </PrivateRoute>
  } />

```

```

10
11 export const PrivateRoute = ({children}) => {
12   const {currentUser} = useContext(UserContext);
13
14   const auth=currentUser?true:false;
15
16   return(
17     <>
18     {auth?children:<Navigate to='/login'/> }
19     </>
20   )
21 }

```

FIGURE-14 : Implementation of Private Routing

4.1.4 Handling Cart Items

Adding to cart is a significant feature of our application, as the user needs to keep track of their selected items. They wouldn't want their saved data to be deleted once they logout. So, we also needed the intervention of the database. So we created another react hooks component named, **CartContext**. This context file will handle the type of items to be added to the cart. As we know, we have four types of providers, it's important to differentiate between them and add to their respective carts for their payment process.

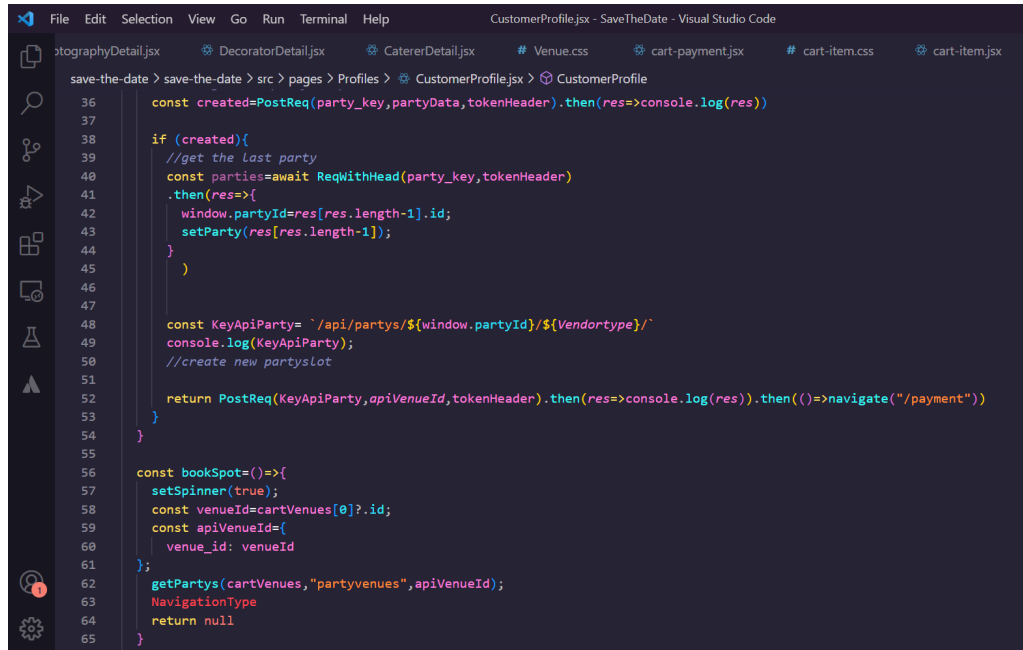


```
File Edit Selection View Go Run Terminal Help cart-context.jsx - SaveTheDate - Visual Studio Code
package-lock.json Pay.jsx Main.jsx cart-context.jsx x ScheduleCard.jsx
save-the-date > save-the-date > src > contexts > cart-context.jsx > addVenue
1 import { createContext, useState } from "react";
2 import React from "react";
3
4 const addCartItem = (cartItems, serviceToAdd) => {
5   const existingCartItem = cartItems.find(
6     (cartItem) => cartItem.id === serviceToAdd.id);
7   if(existingCartItem) {
8     return cartItems.map((cartItem) => cartItem
9   );
10 }
11
12 return [...cartItems, {...serviceToAdd}];
13 }
```

FIGURE-15 : Implementation of Sorting Items into Context

In the backend, there are objects created for specific types of services, upon clicking to checkout, these items give POST requests to the server and add these potential services in the database with a status such as, pending, confirmed with due, confirmed etc. So, once the user logs back in, it simply fetches these data from the server and shows them to their dashboard as they left them. Thus, restoring of the previous phase is done effectively from the front-end using the database.

These cartItems are shown in Booked Items accordion as well.



```
36     const created=PostReq(party_key,partyData,tokenHeader).then(res=>console.log(res))
37
38     if (created){
39         //get the Last party
40         const parties=await ReqWithHead(party_key,tokenHeader)
41         .then(res=>{
42             window.partyId=res[res.length-1].id;
43             setParty(res[res.length-1]);
44         }
45         )
46
47
48         const KeyApiParty= `/api/partys/${window.partyId}/${Vendortype}/`
49         console.log(KeyApiParty);
50         //create new partyslot
51
52         return PostReq(KeyApiParty,apiVenueId,tokenHeader).then(res=>console.log(res)).then(()=>navigate("/payment"))
53     }
54 }
55
56 const bookSpot=()=>{
57     setSpinner(true);
58     const venueId=cartVenues[0]?.id;
59     const apiVenueId={
60         venue_id: venueId
61     };
62     getPartys(cartVenues,"partyvenues",apiVenueId);
63     NavigationType
64     return null
65 }
```

FIGURE-16 : Implementation of Party Booking (CRUD Operation)

4.1.5 Testing

Front-end development is somewhat purely related with the user experience. As much as the user suffers due to server side issues, the client side is highly connected with the user satisfaction. So, it's important for the users to have smooth exchange of the interfaces. Thus, software testing is necessary.

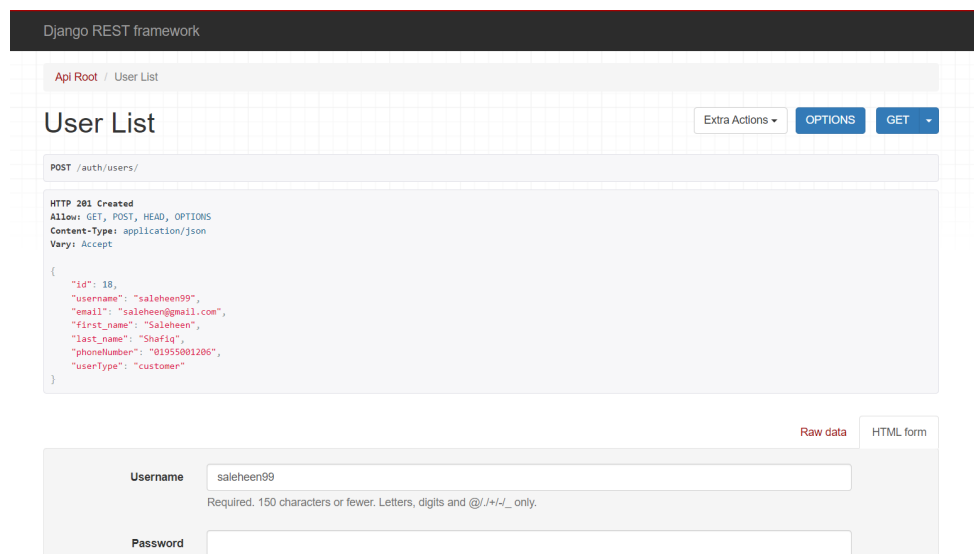
We did unit testing in some components of our application to solve bugs several times. There were some bug fixes related to routing, generating calculative results etc. It appears that our recommendation system needed more valid datas from actual users to be able to provide more efficient results. So, we tested our application with a bunch of dummy datasets to prove its credibility.

4.2 Back-End Implementation

We've used Django as our back-end framework, which is a python language based framework.

4.2.1 Sign Up & Login

The backend of the signup and login system in django was implemented using django's built-in library djoser. The djoser library was used to create a token-based login system. The Django library AbstractUser model contains the following fields, username, password, firstName and lastName. We implemented a Django User Model in our core app which inherits the AbstractUser Model and adds the following fields, email, phoneNumber and userType. There are basic five types of user implemented in the backend. These are customer, venue, catering, content maker and decorator. When a POST request is sent to the url <http://127.0.0.1:8000/auth/users> by filling out the user information a new user is added to the database with the given information.



The screenshot displays the Django REST framework interface for the 'User List' endpoint. The breadcrumb shows 'Api Root / User List'. The endpoint is 'POST /auth/users/'. The response is an HTTP 201 Created status with headers: Allow: GET, POST, HEAD, OPTIONS; Content-Type: application/json; Vary: Accept. The JSON response body is: { "id": 18, "username": "saleheen99", "email": "saleheen@mail.com", "first_name": "Saleheen", "last_name": "Shafiq", "phoneNumber": "01955081206", "userType": "customer" }. Below the response, there are tabs for 'Raw data' and 'HTML form'. The 'HTML form' tab is active, showing a 'Username' field with the value 'saleheen99' and a 'Password' field. A note below the username field states: 'Required. 150 characters or fewer. Letters, digits and @/./+/-_ only.'

FIGURE-17 : Sign Up Completion using REST

While logging in, a post request is sent to the url <http://127.0.0.1:8000/auth/jwt/create> and a refresh and access token is returned as a response. The access token can be used as a request header to gain permissions to access the data related to the user from the backend.

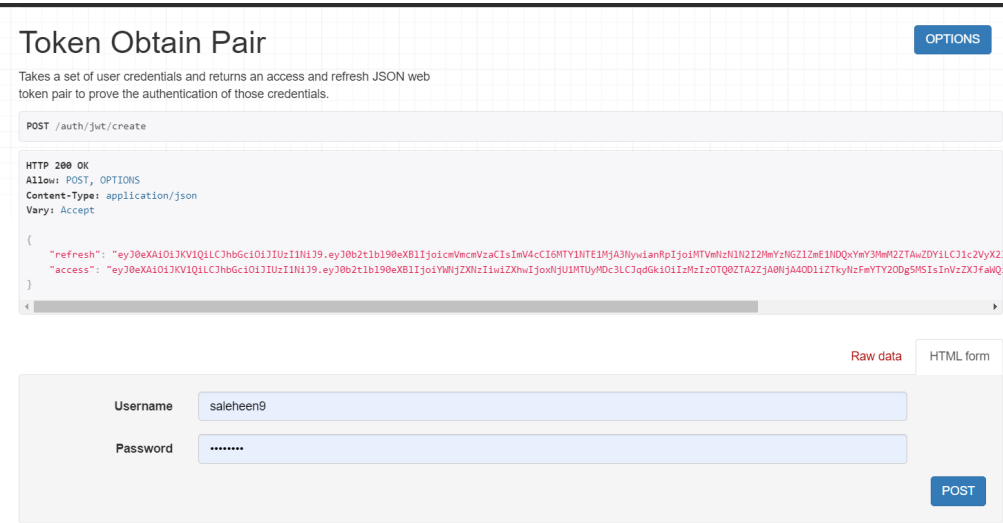


FIGURE-18 : Get Access & Refresh Token

4.2.2 Recommendation System

The recommendation system is mostly handled by the backend of our software. Firstly, a POST request is sent to the url <http://127.0.0.1:8000/api/recommendation> where the recommended service provider's data is returned as Response. In between the request and response, the K-Nearest Algorithm implementation takes place in the recommendation view function.

```

stdBackend > api > views.py > calculatePartyDistance
690
691 @api_view(['GET', 'POST'])
692 def recommendation(request):
693     if request.method=='POST':
694         serializer=RecommendationInputSerializer(data=request.data)
695         serializer.is_valid(raise_exception=True)
696         inputParty={}
697         inputParty['budget']=serializer.validated_data['budget']
698
699         inputParty['locationLatitude']=cityToLongitude(
700             serializer.validated_data['city']
701         )
702         inputParty['locationLongitude']=cityToLongitude(
703             serializer.validated_data['city']
704         )
705         inputParty['guestCount']=serializer.validated_data['guestCount']
706
stdBackend > api > views.py > recommendation
696
697 def calculateMostAppearingServices(partysset, valueOfK):
698     k=valueOfK
699     returnableVenue=0
700     returnableVenueCount=0
701     returnableCatering=0
702     returnableCateringCount=0
703     returnableDecorator=0
704     returnableDecoratorCount=0
705     returnableContentMaker=0
706     returnableContentMakerCount=0
707     venueDictionary={}
708     cateringDictionary={}
709     decoratorDictionary={}
710     contentmakerDictionary={}
711
stdBackend > api > views.py > recommendation
755     if Decorator.objects.filter(id=returnableDecorator).exists():
756         Decoratorquery=Decorator.objects.get(pk=returnableDecorator)
757         Decoratorserializer=DecoratorSerializer(Decoratorquery)
758         decorator.append(Decoratorserializer.data)
759
760     if ContentMaker.objects.filter(id=returnableContentMaker).exists():
761         ContentMakerquery=ContentMaker.objects.get(pk=returnableContentMaker)
762         ContentMakerserializer=ContentMakerserializer(ContentMakerquery)
763         contentmaker.append(ContentMakerserializer.data)
764
765     return Response(
766         {
767             'venue': venue,
768             'catering': catering,
769             'decorator': decorator,
770             'contentmaker': contentmaker
771         }
772     )
773
774
stdBackend > api > views.py > recommendation
705     for i in range(0, min(k, len(partysset))):
706         party=partysset[i]
707         if party.venue_id!=NULL:
708             if party.venue_id in venueDictionary:
709                 venueDictionary[party.venue_id]+=1
710             else:
711                 venueDictionary[party.venue_id]=1
712             if venueDictionary[party.venue_id]>returnableVenueCount:
713                 returnableVenueCount=venueDictionary[party.venue_id]
714                 returnableVenue=party.venue_id
715
716         if party.catering_id!=NULL:
717             if party.catering_id in cateringDictionary:
718                 cateringDictionary[party.catering_id]+=1
719             else:
720                 cateringDictionary[party.catering_id]=1
721             if cateringDictionary[party.catering_id]>returnableCateringCount:
722                 returnableCateringCount=cateringDictionary[party.catering_id]
723                 returnableCatering=party.catering_id
724         if party.decorator_id!=NULL:
725             if party.decorator_id in decoratorDictionary:
726                 decoratorDictionary[party.decorator_id]+=1
727             else:
728                 decoratorDictionary[party.decorator_id]=1
729             if decoratorDictionary[party.decorator_id]>returnableDecoratorCount:
730                 returnableDecoratorCount=decoratorDictionary[party.decorator_id]
731                 returnableDecorator=party.decorator_id
732         if party.contentmaker_id!=NULL:
733             if party.contentmaker_id in contentmakerDictionary:
734                 contentmakerDictionary[party.contentmaker_id]+=1
735             else:
736                 contentmakerDictionary[party.contentmaker_id]=1
737             if contentmakerDictionary[party.contentmaker_id]>returnableContentMakerCount:
738                 returnableContentMakerCount=contentmakerDictionary[party.contentmaker_id]
739                 returnableContentMaker=party.contentmaker_id
740
741     return {
742         'venue': returnableVenue,
743         'catering': returnableCatering,
744         'decorator': returnableDecorator,
745         'contentmaker': returnableContentMaker
746     }
747
748
749     partysset=Party.objects.all().prefetch_related('partyvenueslot')
750
751     partysset=calculatePartyDistance(partysset=partysset, inputParty=inputParty)
752
753     partysset=sorted(partysset, key=lambda x:x.distance)
754     serializer=PartySerializer(partysset, many=True)
755     return calculateMostAppearingServices(partysset)
756

```

FIGURE-19 : Recommendation Models

First all the data related to previous party transactions are retrieved using object relational mapper in Django. The party queryset is iterated and then distance is calculated using the given parameters of the customer and the similar parameters of each party. The distance is calculated in the Euclidean method and different weights are assigned to calculate differences between the parameters. Thus the parameters are normalized to calculate the distance of the parties from the user provided parameters. The party objects are then sorted based on their distance with the provided parameters. From the sorted list of party objects, the first k objects are iterated where we are using k=20 in our system. The services that are appearing more in the first k party objects will be recommended to the customer.

4.2.3 Party Booking API

A party object in the backend can be created using the url <http://127.0.0.1:8000/api/partys>. A POST request in the url with the value of guestCount will create a new party object with an empty status. The party venues, caterings, content makers and decorators can then later be added using extended urls. For instance, to book a venue http://127.0.0.1:8000/api/partys/<party_id>/partyvenues url will be used. The venue_id of any particular venue provider will be posted in the url to book the certain venue. Similarly the party venue can be updated using PUT request by adding the partyvenue_id extension after the url.

Service To Book	URL	Values to post
Venue	<a href="http://127.0.0.1:8000/api/partys/<party_id>/partyvenues">http://127.0.0.1:8000/api/partys/<party_id>/partyvenues	venue_id
Catering	<a href="http://127.0.0.1:8000/api/partys/<party_id>/partycaterings">http://127.0.0.1:8000/api/partys/<party_id>/partycaterings	catering_id foditem_id
Decorator	<a href="http://127.0.0.1:8000/api/partys/<party_id>/partydecorators">http://127.0.0.1:8000/api/partys/<party_id>/partydecorators	decorator_id
Photographer	<a href="http://127.0.0.1:8000/api/partys/<party_id>/partycontentmakers">http://127.0.0.1:8000/api/partys/<party_id>/partycontentmakers	contentmaker_id

TABLE-2 : URL Distribution for Party API

At first, the Party model was used to design the structure to the Party object in which it will be stored in the database. The PartyViewSet Class was implemented to

retrieve all the party objects from the database. The class uses serializers to specify the data that will be communicated with the frontend. In the serializer, the `paidPrice` method was used to calculate the total paid price from all the payment objects in the database related to that specific party. Party object has one to one relationship with the following classes: `PartyVenue`, `PartyCatering`, `PartyDecorator` and `PartyContentMaker`. Each of them uses a separate viewset to extend the url and add a new endpoint for adding and updating services to the database.

```

78
79
80 class Party(models.Model):
81     customer=models.ForeignKey(Customer, on_delete=models.CASCADE)
82     catering=models.ManyToManyField(Catering)
83     totalCost=models.DecimalField(
84         max_digits=11,
85         decimal_places=2,
86         default=0,
87     )
88     status=models.CharField(max_length=255, default="unconfirmed")
89     locationLatitude=models.DecimalField(max_digits=5, decimal_places=2, default=Decimal(23.8103))
90     locationLongitude=models.DecimalField(max_digits=5, decimal_places=2, default=Decimal(98.4125))
91     guestCount=models.IntegerField(default=0,validators=[MinValueValidator(0)])
92
93     @property
94     def get_totalCost(self, party):
95         try:
96             totalCost=(sum([partyvenueslot.venueslot.price for partyvenueslot in party.partyvenueslot.all()])
97             +sum([cartitem.fooditem.unitPrice*cartitem.quantity for cartitem in party.foodcartitem.all()])
98             +sum([partytheseslot.themeslot.price for partytheseslot in party.partytheseslot.all()]) )
99         except:
100             self.totalCost=totalCost
101             return totalCost
102         except AttributeError:
103             return 0
104

```

```

461
462 class PartySerializer(serializers.ModelSerializer):
463     foodcartitem=FoodCartitemSerializer(many=True, read_only=True)
464     partyvenue=PartyVenueSerializer(many=True, read_only=True)
465     totalCost=serializers.SerializerMethodField()
466     partydecorator=PartyDecoratorSerializer(many=True, read_only=True)
467     partycontentmaker=PartyContentMakerSerializer(many=True, read_only=True)
468     payedPrice=serializers.SerializerMethodField()
469     pendingPrice=serializers.SerializerMethodField()
470     status=serializers.SerializerMethodField()
471
472     class Meta:
473         model=Party
474         fields=('id', 'totalCost', 'payedPrice', 'pendingPrice',
475             'status', 'foodcartitem', 'partyvenue', 'partydecorator', 'partycontentmaker',
476             'guestCount', 'locationLatitude',
477             'locationLongitude')
478
479     def get_payedPrice(self, party):
480         return sum([payment.amount for payment in party.payment.all()])
481
482
483     def get_totalCost(self, party):
484         totalCost=(sum([partyvenue.venue.price for partyvenue in party.partyvenue.all()])
485         +sum([cartitem.fooditem.unitPrice*cartitem.quantity for cartitem in party.foodcartitem.all()])
486         +sum([partydecorator.decorator.price for partydecorator in party.partydecorator.all()])
487         +sum([partycontentmaker.contentmaker.price for partycontentmaker in party.partycontentmaker.all()])
488         )
489

```

```

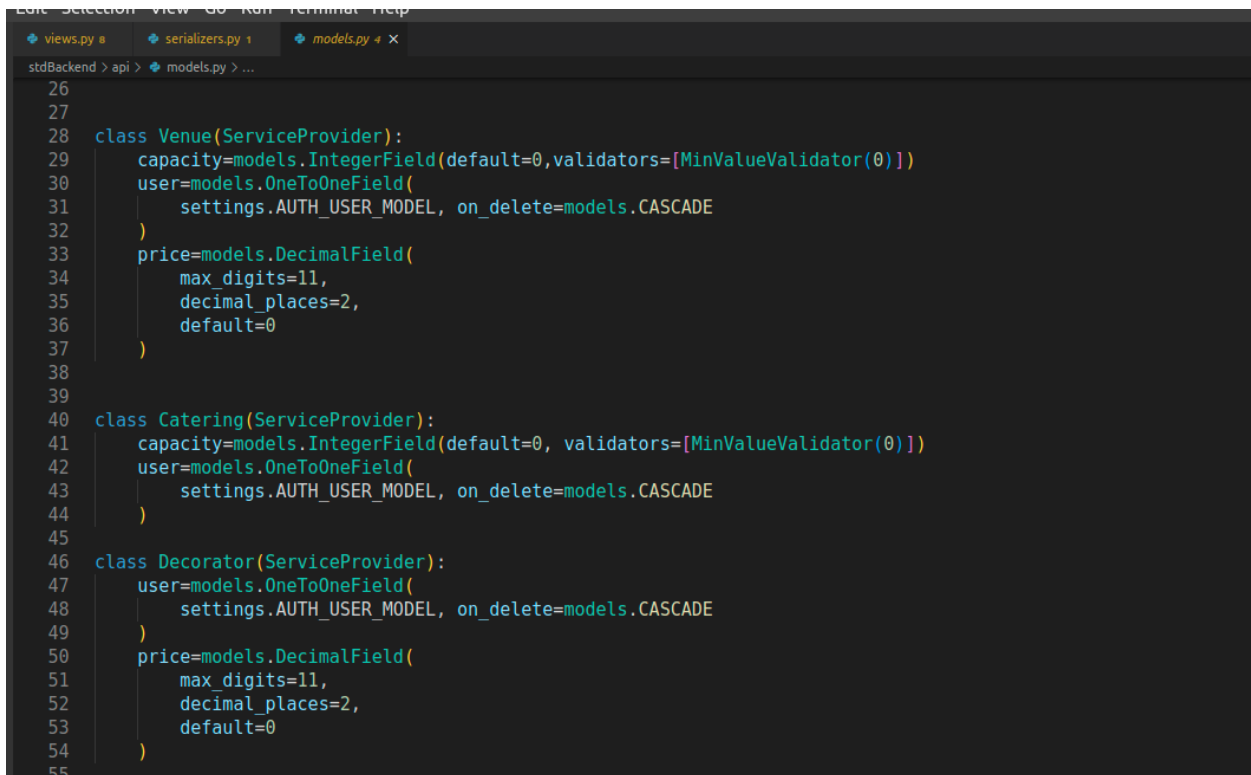
363
364
365 class PartyViewSet(ModelViewSet):
366     def get_queryset(self):
367         user = self.request.user
368
369         if user.is_staff:
370             return Party.objects.all()
371
372         if user.userType=="customer":
373             customer_id = Customer.objects.only(
374                 'id').get(user_id=user.id)
375             return Party.objects.filter(customer_id=customer_id)
376
377         elif user.userType=="venue":
378             partyset=Party.objects.all()
379             returnablePartySet=partyset.none()
380             for party in partyset:
381                 if PartyVenue.objects.filter(party_id=party.id).exists():
382                     partyvenue=PartyVenue.objects.get(party_id=party.id)
383                     venue=Venue.objects.get(id=partyvenue.venue_id)
384                     party.venue_id=Venue.objects.get(id=venue.id).id
385                     returnablePartySet|=Party.objects.filter(pk=party.id)
386
387             return returnablePartySet

```

FIGURE-20 : Party API Models

4.2.4 Profile Update

A profile will be automatically generated based on the userType field which the user filled while registering. If the registered user is a customer, the profile settings can send GET requests to <http://127.0.0.1:8000/api/customers/<id>> to receive customer profile info as response. The frontend can send PUT requests to the same url to update the info of the customers. For service providers, if the user is a venue provider the system will communicate with the backend similarly using the url <http://127.0.0.1:8000/api/venues/<id>>.



```
26
27
28 class Venue(ServiceProvider):
29     capacity=models.IntegerField(default=0,validators=[MinValueValidator(0)])
30     user=models.OneToOneField(
31         settings.AUTH_USER_MODEL, on_delete=models.CASCADE
32     )
33     price=models.DecimalField(
34         max_digits=11,
35         decimal_places=2,
36         default=0
37     )
38
39
40 class Catering(ServiceProvider):
41     capacity=models.IntegerField(default=0, validators=[MinValueValidator(0)])
42     user=models.OneToOneField(
43         settings.AUTH_USER_MODEL, on_delete=models.CASCADE
44     )
45
46 class Decorator(ServiceProvider):
47     user=models.OneToOneField(
48         settings.AUTH_USER_MODEL, on_delete=models.CASCADE
49     )
50     price=models.DecimalField(
51         max_digits=11,
52         decimal_places=2,
53         default=0
54     )
55
```

FIGURE-21 : Profile API Models

User Type	URL
Customer	http://127.0.0.1:8000/api/customers/<id>
Venue	http://127.0.0.1:8000/api/venues/<id>
Catering	http://127.0.0.1:8000/api/caterings/<id>
Decorator	http://127.0.0.1:8000/api/decorators/<id>
Photographer	http://127.0.0.1:8000/api/contentmakers/<id>

TABLE-3 : URL Distribution for Profile API

The user can add images to their gallery using /images extension to the profile urls. Images can be added to the backend using POST request and also received from the backend using the GET request. The catering service provider can add different food items using the URL <http://127.0.0.1:8000/api/caterings/<id>/items> where food items can be added using POST request and the data can of food item can be received by GET request.

The models for different kinds of users were built to organize the user data in the database. The viewsets use models to retrieve the data from the database. The viewsets to particular users also use serializers to determine how the data will be communicated with the backend.

CHAPTER 5

USER MANUAL of Save the date

Our user interface will have a homepage to start with like every other web application. The homepage will have the following-

- Navigation Bar
- Cover Page
- Exploration Section

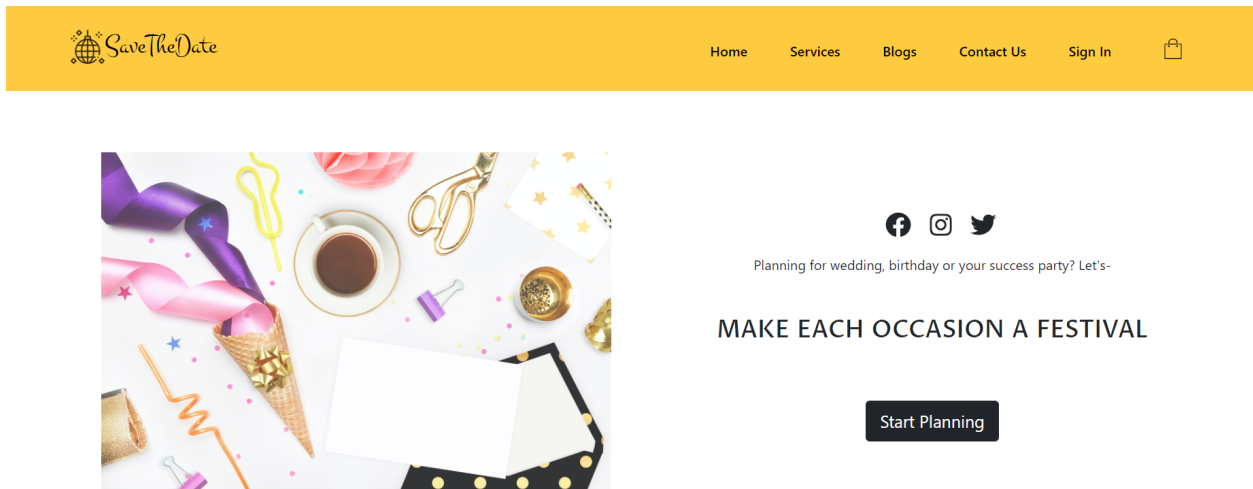


FIGURE-22 : HomePage of Save the date

5.1 Authentication & Authorization

In the navigation bar, there's a sign in button. Upon clicking the button, it'll take us to the sign in/register page. If we already have an account, we can simply provide our username and password. If the password is incorrect, it may show us an alert containing the message. Alternatively, if we wish to open an account, we can

traverse to the Register section. There we need to provide necessary information, upon required fields being filled, we'll be able to successfully register ourselves as a user.

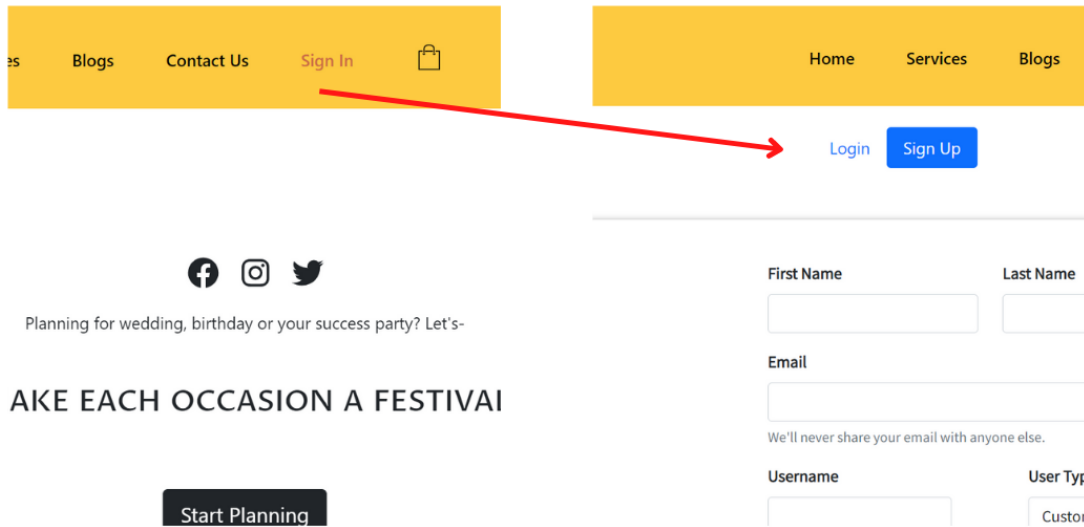


FIGURE-23 : Sign In to Login/Register Page

5.2 Navigation Bar

Our navigation bar consists of five items-

Navigation Item Name	Routed Path Functionality
Home	Take us to the homepage.
Services	Help us to explore all kinds of services.
Blogs	Show us all uploaded blog contents.
Contact	Submit the feedback form.
Sign In/Logout (Profile Icon show)	Login/Register page (Profile if logged in)

TABLE-4 : Navigation Bar Routings

5.3 Customer View

If userType given is customer and logs in, there will be dashboard shown which will have following functionalities-

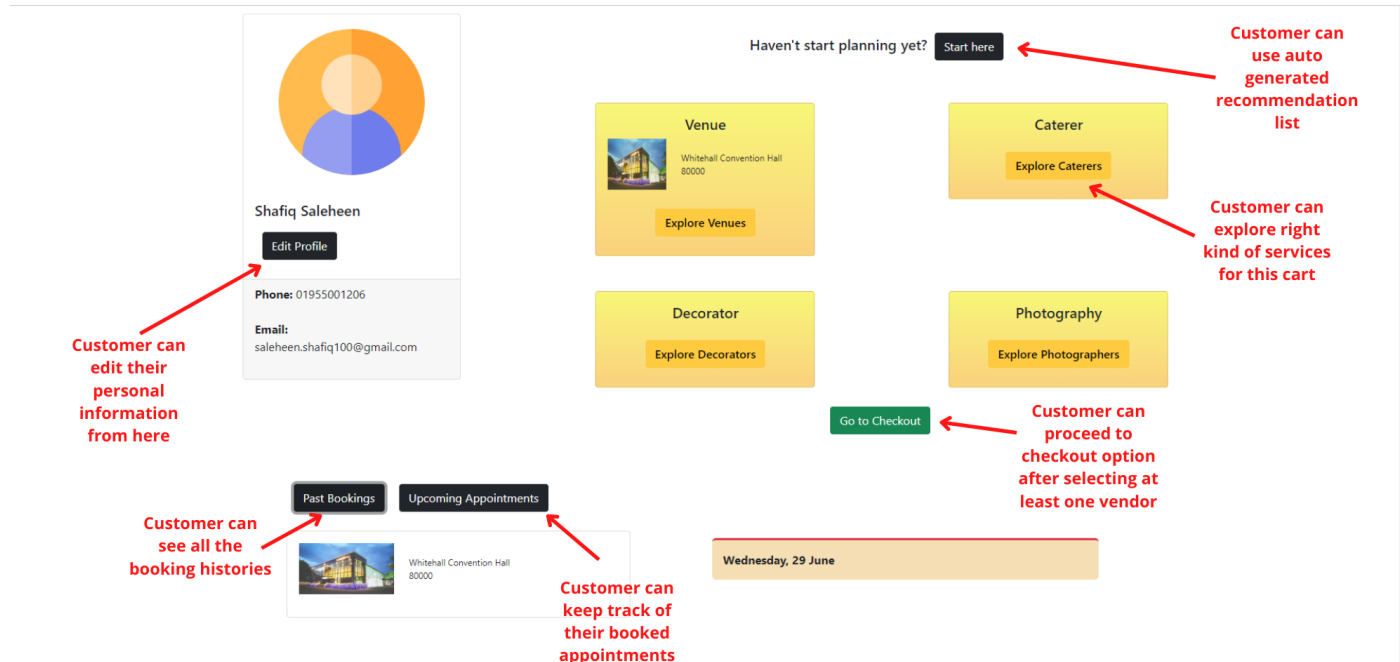


FIGURE-24 : Customer Dashboard

A user can explore for services, go to recommendations, book appointments, update schedules, edit profiles etc. from their customer dashboard. The 'Go to Checkout' button is disabled until at least one item is selected.

5.3.1 Booking Section

The selected items in the carts will have a status attribute which will show the current situation of the booking. It may show 'pending' which means- vendor booked, but haven't paid yet. It will show 'confirmed' status if we complete full payment, otherwise it will show the due amount. If payment is successful, a booking object will be placed in the dashboard of the customer.

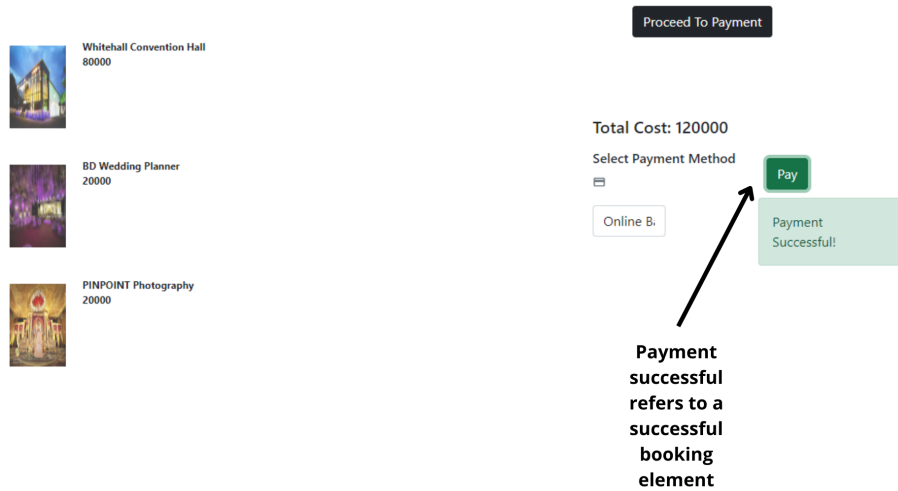











FIGURE-25 : Payment Section Manual

5.3.2 Food Item Selection

There's a separate section in every caterer's modal view which will enable the customers to select items and then add to cart.

Item	Name	Description	Quantity	Price	Remove
	Chicken Biryani	Chicken Biryani Is A Delicious Savory Rice Dish Loaded With Spicy Marinated Chicken, Caramelized Onions, And Flavorful Saffron Rice.	<input type="text" value="1"/>	300	
	Lasagna	Sinfully Rich And Packed With Flavors. Arguably One Of The Best Lasagnas In Town	<input type="text"/>	260	
	Nargisi Kofta	Boneless Chicken Molded By Hand In Fresh Herbs And Spices. Served In A Curry Sauce	<input type="text"/>	60	
	Beef Kala Bhuna	Fresh Undercut Boneless Beef Mixed With Traditional Spices And Cooked To Perfection Till It Simply Melts In The Mouth	<input type="text"/>	180	
	Water & Borhani Jug	For Water And Borhani; 1.5 Liter Capacity; Serves ~8 People	<input type="text"/>	80	

Total: 0

FIGURE-26 : Food Selection Manual

5.3.3 Recommendation System

If we go to start planning, the system will generate an UI to have its input parameters to start with the recommendation system. The very first page will accept the type of party we would like to arrange. Then, we would need to provide the preferred date, city location, area location, approximate budget and approximate guest counts.

Using server side, it'll collect a full recommendation plan for the customer which will suit their budget, location, accommodation and everything else.

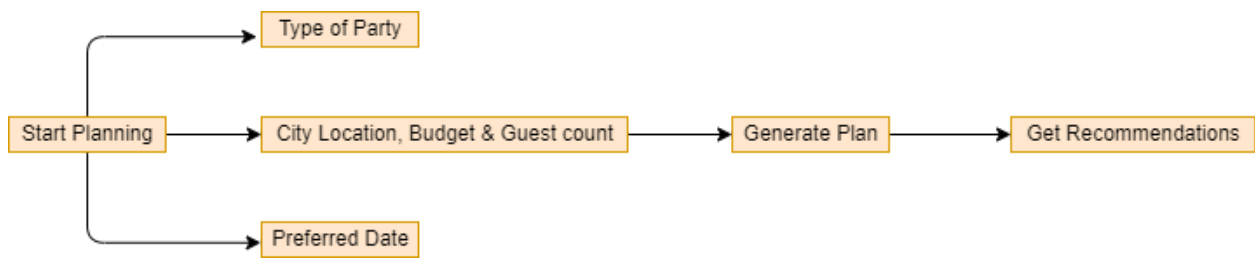


FIGURE-27 : Diagram of Recommendation Manual

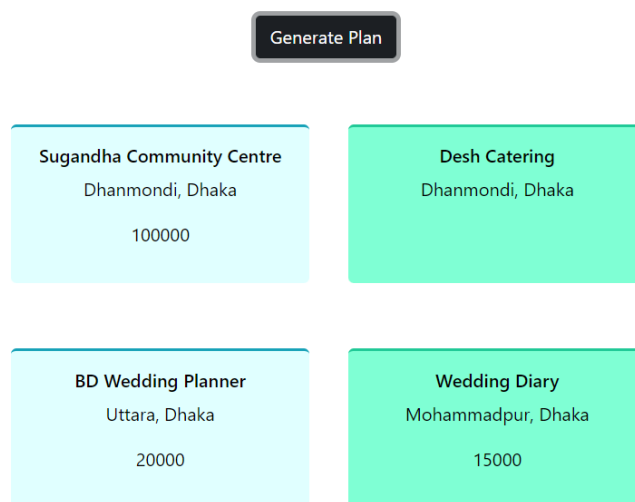


FIGURE-28 : Generated Plan for the Customer

5.4 Provider View

If userType is selected among Venue/Caterer/Decorator/Photographer, the user will be redirected to a profile view where they can showcase their services to the customers-

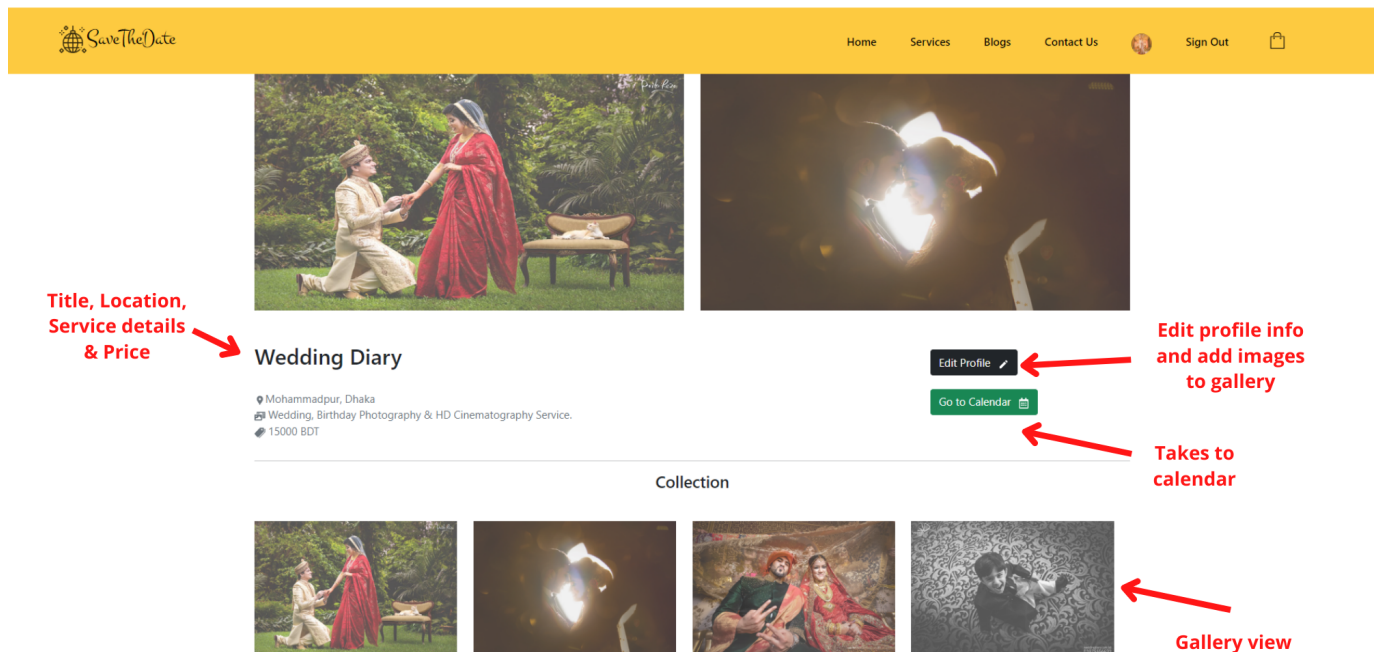


FIGURE-29 : Provider Profile Page Manual

There's a calendar view in the provider's profile, where they can see their scheduled appointments. This also notifies the user on the day of the event.

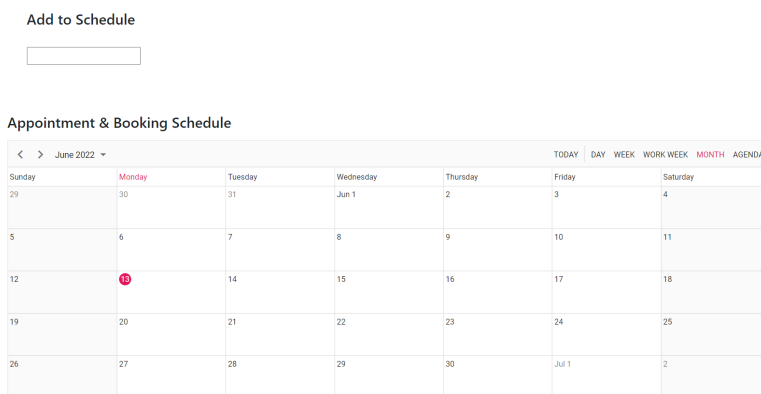


FIGURE-30 : Calendar

The providers can also edit their profiles from a separate page where their information will be displayed. They can edit them as they see fit, also can drop images in the gallery section to view.

Location Where's the venue situated?

Capacity Approx. number of guests?

Title Venue name?

Description Details of the venue?

Choose File No file chosen

Save

s

FIGURE-31 : Edit Profile Page

This user manual covers every section of our website. It gives us a clear path through the application to surf as easily as possible for the customer and service provider.

Here's a video link given below to see the full user manual-

[SaveTheDate- Demo Video for User Manual](#)

CHAPTER 6

CHALLENGES FACED in implementing

As software engineering students, we had prior knowledge about web technologies. Although, the concept of three-tier architecture and frameworks were something that we had to learn upon during the project timeline. Few of the challenges we faced during our three-month project were-

6.1 Learning to Collaborate & Distribute

As we've followed the three-tier architecture, it was given that we've to maintain a front-end or client side with a back-end or server side along with an additional database management system. To follow through the technologies, we had to brainstorm and derive a plan to do research, background study and finally coming up with appropriate technology that may go with our plan were all a bit hectic. But as soon as we derived our general plan, we were able to catch up with previous stallments.

6.2 Requirements Engineering & Data Collection

To build a product, finding its requirements and specifications, are probably the most primary thing to do. However, in our case, we didn't rest only by collecting requirements. Our application is heavily based on real-time data as we've a recommendation system which is built upon past actions. So, we had to go from vendor to vendor to collect real data. Again, our system had two types of users and five types of sub users. So, acting upon different scenarios was a real difficult thing to do while doing requirement analysis.

However, we managed to collect enough datasets, but also needed to rely on dummy datasets as well for testing purposes.

6.3 Version Updates

While implementing the front-end, React JS has different component structures to make things happen. One of them is- react-router-dom. In our case, what happened is, in React JS version 18, there was a significant update in the syntax. In the react-router-dom library, a syntax change happened from v5 to v6. V6 is only runnable from React JS 18. But the learning process we were following had documentation of React JS versions lower than 17. So, every time we loaded, it returned an error and a blank page because of the react-router-dom v5 version. Similar problems happened with multiple libraries in both frontend and backend. This was a drawback for us. However, we learned from mistakes and looked for the appropriate documentation from then on.

<pre>6 function App() { 7 return (8 <Switch> 9 <Route exact path="/" component={Home} /> 10 <Route path="/about" component={About} /> 11 <Route 12 path="/users/:id" 13 render={({ match }) => (14 <User id={match.params.id} /> 15)} 16 /> 17 </Switch> 18); 19 } 20</pre>	<pre>2 import { 3 BrowserRouter, 4 Routes, 5 Route, 6 Link, 7 } from "react-router-dom"; 8 9 function App() { 10 return (11 <BrowserRouter> 12 <Routes> 13 <Route path="/" element={Home} /> /> 14 <Route path="users/*" element={Users} /> /> 15 </Routes> 16 </BrowserRouter> 17); 18 }</pre>
BEFORE V17	AFTER V17

FIGURE-32 : Syntax Changes of react-router-dom ("Upgrading from v5")

6.4 CRUD Operations

We used React JS as front-end and Django framework as backend. React is a JavaScript based language framework, whereas, Django has Python language as its implementation criteria. So, at the end of the day, while performing CRUD

operations, we faced a bit of difficulty as GET, POST, PUT methods have different types of implementations in both languages.

To overcome the problem, we had to study a bit more and then we generated a set of functions that could easily translate the methods. Thus, we connected the front-end to the back-end.

6.5 Back-end Complications

One of the key challenges faced during the development of the project was to learn the Django framework which was used to build the backend. The Django framework uses models to build database structure, serializers to communicate data with the frontend and it uses views as methods and viewsets as classes which were difficult to comprehend from scratch.

6.6 Recommendation Parameter Settings

Another key challenge was building the recommendation system using the K-Nearest Neighbor algorithm. At first, determining the right algorithm for the system required a lot of research and seeking expert advice. The K-Nearest Algorithm was applied in our project where we had to collect more than 100 party transaction data and use it to apply the K-Nearest Algorithm. It is worth mentioning that the attributes used for K-Nearest Neighbor Algorithm needed to be normalized before application where the best normalization technique was determined using trial and error method.

CHAPTER 7

FUTURE WORK & CONCLUSION of Save the date

7.1 Future Work

Our web based application has a strong basis to establish it as an independent product in the market. To do so, there's a few features we would like to add-

- A live progress tracking system, where vendors update will be automatically notified to the host of the event.
- A chatbot to eradicate the gap to an extent between the customer and the provider.
- More customization features can be added, as long as the providers are interested. It may cause customers to high up their interest in these services.
- Vendors may have the opportunity to promote their services through-campaigns, offers, promo codes etc.
- Maintaining a guest-list would be beneficial for the host to keep updated with all the people coming and also happen to be helpful in case of collecting gifts and note them as well.
- We can implement a feature where we can create automatically generated cards for party invitations and auto email feature,
- A mobile application using React Native could be a key to sell the product hand-to-hand in every corner of the world.
- A proper business model should be developed to promote its cost effectiveness. Thus, it can get necessary fundings from big corporations, or it can also start its journey as an entrepreneur project.

7.2 Conclusion

So, 'SaveTheDate' is a party-planning and booking services application which provides an all-in-one service recommendation for the hosts to save time and reduce hassle. It includes booking a venue, hiring catering services, decorators or designers etc. Using modern technology, this application will recommend service providers based on their location, budget and availability. The generation of an automated plan and the customization properties will help the planner to construct a successful event with more flexibility and less struggle.

This project helped us achieve new skills in the field of web technologies and polished us for our path further in web development. We have learned a lot about how to develop a software, how to gather and analyze the requirements, the software designing process and finally the implementation phase. Documentation has been a key process in our project. We developed a SRS document and finally understood the application of it. We're hopeful that our learnt technologies will help us in the long run, in software industries to flourish ourselves as good software developers.

REFERENCES

[Django Usefulness](#)

[REST API](#)

[React Router Version Updates](#)

[Three Tier Architecture](#)

[Top Reasons to Use MySQL \(databasequest.com\)](#)