Table of content

INTRODUCTION

Topic

The **Catering Services Booking Application** is a web-based platform aimed at streamlining the booking and management of catering services for events such as weddings and birthday parties. It allows users to view available event packages, make reservations, and communicate with the service provider via a contact form. The application includes secure user authentication and administrative tools for managing event offerings and customer inquiries.

Project Purpose

The primary objective of this engineering project is to design and develop a modern, web-based application that simplifies the process of booking catering services. The system aims to eliminate the inefficiencies of traditional manual methods by providing an intuitive interface for customers to select their desired event type, specify their location and number of guests, and finalize bookings. At the same time, the platform equips administrators with tools to manage event packages and monitor client communications effectively.

Scope

The current scope of this project encompasses the following event types:

- Birthday Parties
- Weddings

Key features include:

- A user-friendly event booking interface
- The ability to select either predefined or custom locations
- Guest number management
- An administrative dashboard for managing event packages and client inquiries

1. THEORY OF ONLINE CATERING BOOKING SYSTEMS

1.1 Introduction

Catering is a critical component of event planning, involving the coordination of food preparation, service, and logistical support. Traditionally, clients arranged catering through phone calls or physical visits to vendors, a process that is often time-consuming and error-prone. These manual systems lack real-time availability, transparency, and scalability.
In response to these challenges, online catering booking systems have emerged. These platforms utilize modern web technologies to enable clients to book services online, view available options instantly, and receive confirmation with minimal delay. As a result, the catering industry has experienced a shift toward digital solutions that provide greater convenience, accuracy, and customer satisfaction.

1.2 The Need for an Online Catering Booking System

Manual booking systems introduce various challenges:

- **Limited Accessibility:** Clients must contact providers during business hours or visit physical locations.
- **Scheduling Conflicts:** Double bookings and miscommunications are common with handwritten schedules or spreadsheets.
- **Lack of Transparency:** Clients cannot easily compare packages, prices, or availability.
- **High Administrative Overhead:** Staff must manually record, confirm, and update bookings.

A web-based catering booking system addresses these issues by:

- **Automating the Booking Process:** Clients can complete bookings anytime, anywhere.
- **Reducing Human Error:** Real-time data validation and confirmation improve

reliability.

- **Customizing Event Details:** Users can choose event types, locations, dates, and guest numbers
- **Enabling Administrative Control:** Admins can manage packages, view upcoming bookings, and respond to inquiries.
- **Improving Customer Experience:** Users enjoy a smoother, more professional experience when interacting with a digital platform.

1.3 Technologies Used

The system architecture is designed with simplicity, scalability, and security in mind. The core technologies used include:

- **Back-End:**
    - **Flask (Python):** A lightweight web framework used to build the server-side logic.
    - **SQLite:** A self-contained database engine suitable for lightweight applications and development.
- **Front-End:**
    - **HTML & CSS:** Structure and styling of the web pages.
    - **JavaScript & AJAX:** Dynamic content updates and asynchronous server communication.
- **Security & Authentication:**
    - **Flask-Login:** Manages user session handling and login functionality.
    - **Werkzeug Security:** Provides password hashing and secure session handling.

1.4 Comparison with Other Booking Systems

Compared to other online service booking systems (e.g., travel or salon appointments), catering booking platforms must support greater customization due to the complexity of events. For instance, a catering event may vary based on:

- **Menu Selection and Dietary Restrictions**
- **Venue Type and Layout Needs**
- **Timing and Staffing Requirements**

Therefore, a catering booking system must offer a high degree of flexibility and real-time administrative control to accommodate diverse client needs.

1.5 Benefits to Stakeholders

For Clients:

- 24/7 booking convenience
- Transparent pricing and package options
- Faster confirmation and less room for error

For Service Providers:

- Reduced administrative burden
- Better organization and scheduling
- Ability to gather analytics for business growth

2. PROJECT OF THE CATERING SERVICES BOOKING APPLICATION

To develop reliable and user-friendly software, it is essential to begin with a detailed design phase. This involves understanding the system's goals, identifying how it will function, and specifying what is needed to build and support it. Proper software design ensures that the final product is scalable, secure, and maintainable.

This chapter outlines the core requirements of the system, grouped into functional, non-functional, and hardware categories.

2.1 Project Requirements

Functional Requirements

These specify the system's core functions and behaviors:

1. **User Registration and Login**
   - Users must be able to register with a unique username and email.
   - Passwords must be stored securely using hashing algorithms.
   - Logged-in users should maintain sessions until logout.
2. **Role-Based Access Control**
   - Only authenticated users can access booking and history pages.
   - Admin users have access to package management and inquiry views.
3. **Package Viewing and Filtering**
   - All users can view available catering packages.
   - Packages can be filtered by event type (wedding, birthday).

- Each package includes detailed descriptions, pricing, and guest limits.

4. **Event Booking System**
   - Authenticated users can create bookings by selecting a package and entering event-specific information (date, location, guest count).
   - The system should confirm and store the booking upon successful submission.

5. **Booking Management**
   - Users can view their booking history and upcoming events.

6. **Inquiry Submission**
   - Users (guests or logged-in) can submit inquiries through a contact form.
   - Admins can view submitted inquiries from the admin dashboard.

7. **Validation and Feedback**
   - All form fields (registration, booking, inquiry) should include validation (e.g., required fields, formats).
   - The system must display user-friendly error messages for failed action

Non-Functional Requirements

These describe how the system performs under various conditions:

1. **Security**
   - Passwords must be hashed using Werkzeug
   - Sessions should be protected against hijacking.
   - HTTPS should be enforced in production.

2. **Performance**
    - The system should support at least 20–50 concurrent users without significant latency.
3. **Usability**
    - Interfaces must be clean, responsive, and intuitive for both desktop and mobile users.
4. **Reliability**
    - The application should be consistently available with 99% uptime.
    - Booking data should be persisted even after server restarts.
5. **Data Integrity**
    - Bookings, user accounts, and inquiries must be stored accurately and consistently in the database.
6. **Cross-Platform Compatibility**
    - The application must function correctly on major browsers (Chrome, Firefox, Edge, Safari).
    - Layout and features must adapt to various screen sizes.
7. **Scalability**
    - The architecture should allow easy extension to new event types, features (e.g., online payments), or third-party integrations.

Hardware Requirements

These are the minimum hardware specifications needed for development and deployment:

Development Environment

- **Processor:** Intel Core i5 or AMD equivalent
- **Memory (RAM):** 8 GB minimum
- **Storage:** 256 GB SSD or higher
- **Operating System:** Windows 10+, Linux, or macOS
- **Internet:** Stable connection for development, version control, and testing

Deployment Environment (Server)

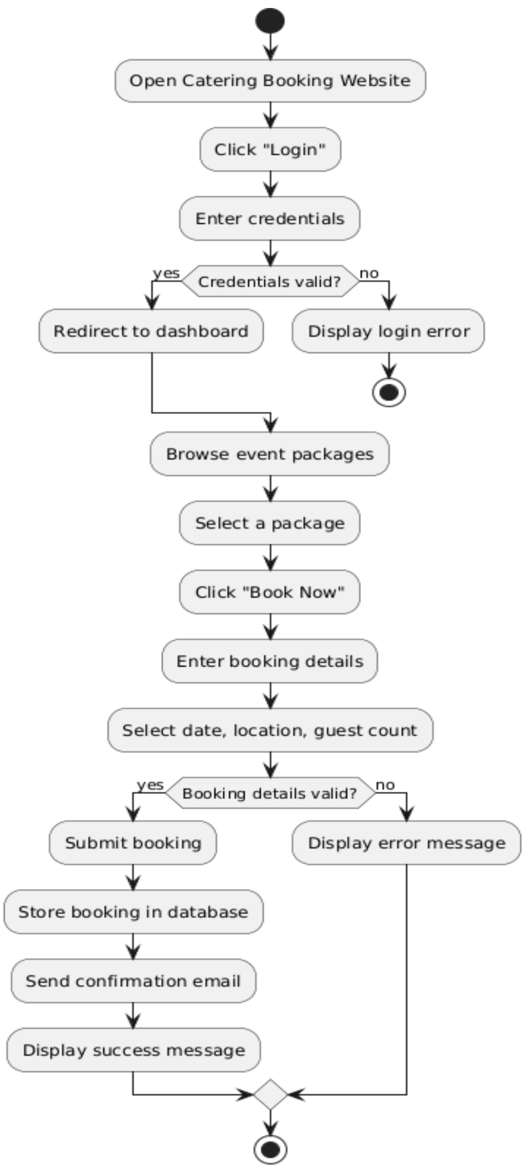
- **Processor:** Dual-core (2.4GHz or above)
- **Memory:** 4 GB RAM minimum (8 GB recommended)
- **Storage:** 50 GB minimum for app, logs, and DB
- **Operating System:** Linux-based Ubuntu Server
- **Web Server:** Nginx or Apache
- **Database:** SQLite (local development), PostgreSQL/MySQL (scalable deployment)

2.2 UML Diagrams

To effectively model and communicate the design of the Catering Services Booking Application, various UML diagrams were created. These diagrams illustrate both the structural and behavioral aspects of the system, helping to clarify system functionality, user interactions, and internal processes.

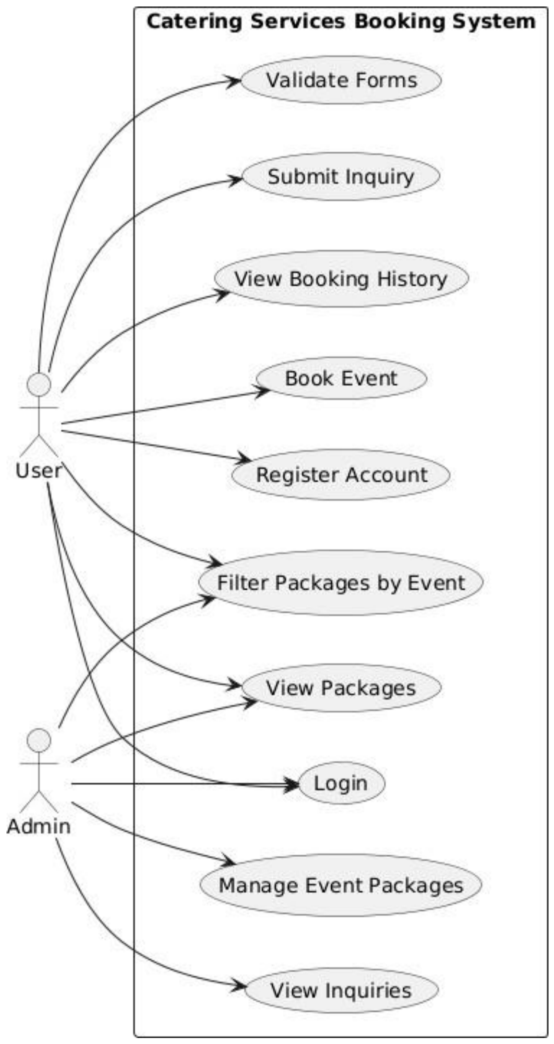
2.2.1 Activity Diagram

Models the step-by-step process of booking an event, from login to confirmation. It visualizes the logical flow and decision points in the booking process.



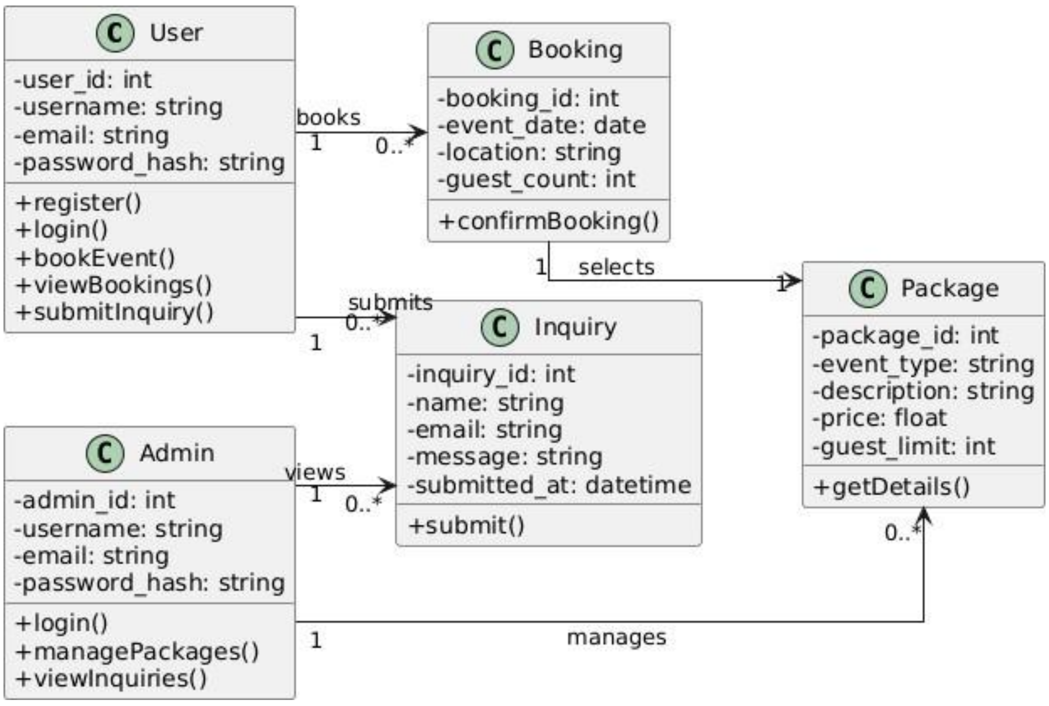
2.2.2 Use case diagram

This diagram shows how Users and Admins interact with the Catering Services Booking System. Users can register, book events, view packages, and submit inquiries. Admins manage event packages and respond to inquiries, while also accessing shared features like login and package viewing.



2.2.3 Class diagram

The class diagram represents the core structure of the Catering Services Booking System. It defines key classes such as User, Admin, Package, Booking, and Inquiry, along with their attributes and methods. Relationships between classes like associations between users and bookings highlight how data is organized and managed within the system.

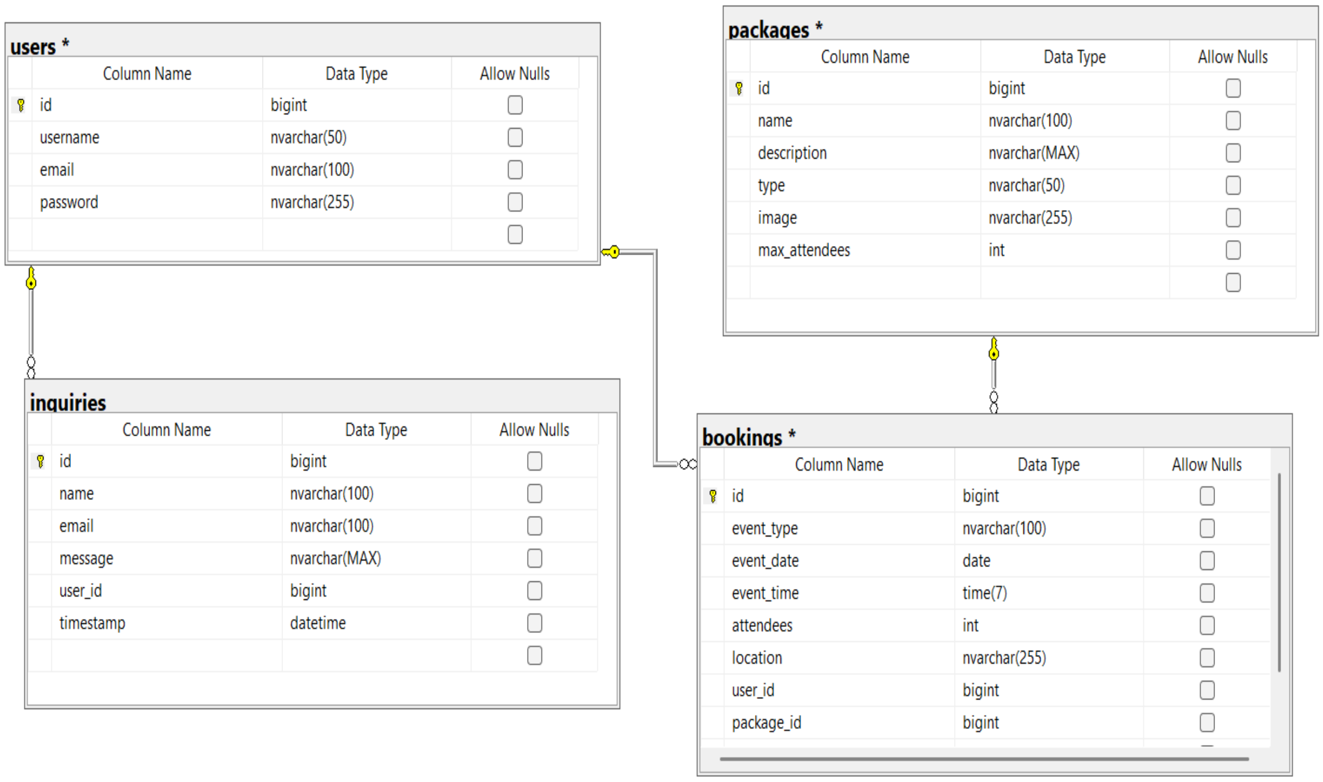


2.3 Database diagram

The database for the Catering Services Booking Application is designed to efficiently handle and organize all essential data related to users, bookings, packages, and inquiries. It supports key functionalities such as event booking, user authentication, and customer support.

- **Users**: Stores registered user credentials and contact information, enabling authentication and personalized bookings.
- **Bookings**: Tracks user reservations, including event details like date, time, location, and the selected package.
- **Packages**: Maintains available catering offerings, each described by a name, type, and optional image.
- **Inquiries**: Captures messages submitted through the contact form, linking them to users where applicable for follow-up.

This relational design ensures data consistency, supports smooth user experiences, and allows for secure access control and administrative management.



users *

Column Name	Data Type	Allow Nulls
id	bigint	☐
username	nvarchar(50)	☐
email	nvarchar(100)	☐
password	nvarchar(255)	☐
		☐

packages *

Column Name	Data Type	Allow Nulls
id	bigint	☐
name	nvarchar(100)	☐
description	nvarchar(MAX)	☐
type	nvarchar(50)	☐
image	nvarchar(255)	☐
max_attendees	int	☐
		☐

inquiries

Column Name	Data Type	Allow Nulls
id	bigint	☐
name	nvarchar(100)	☐
email	nvarchar(100)	☐
message	nvarchar(MAX)	☐
user_id	bigint	☐
timestamp	datetime	☐
		☐

bookings *

Column Name	Data Type	Allow Nulls
id	bigint	☐
event_type	nvarchar(100)	☐
event_date	date	☐
event_time	time(7)	☐
attendees	int	☐
location	nvarchar(255)	☐
user_id	bigint	☐
package_id	bigint	☐

Table 1: users – fields and data types

Field Name	Data Type	Attributes	Description
id	long	Primary key	Unique identifier for each user
Username	nvarchar(50)	not null	The display name used for login
Email	nvarchar(50)	not null	User's email address
Password	nvarchar(50)	not null	Hashed password for authentication

Table 2: inquiries – fields and data types

Field Name	Data Type	Attributes	Description
id	long	Primary key	Unique identifier for each inquiry
name	nvarchar(50)	not null	Name of the person submitting the inquiry
email	nvarchar(50)	not null	Email address of the person
message	nvarchar(200)	not null	The content/message of the inquiry
userID	int	Foreign key, nullable	Reference to registered user
timestamp	datetime	not null	Date and time when the inquiry was submitted

Table 3: packages – fields and data types

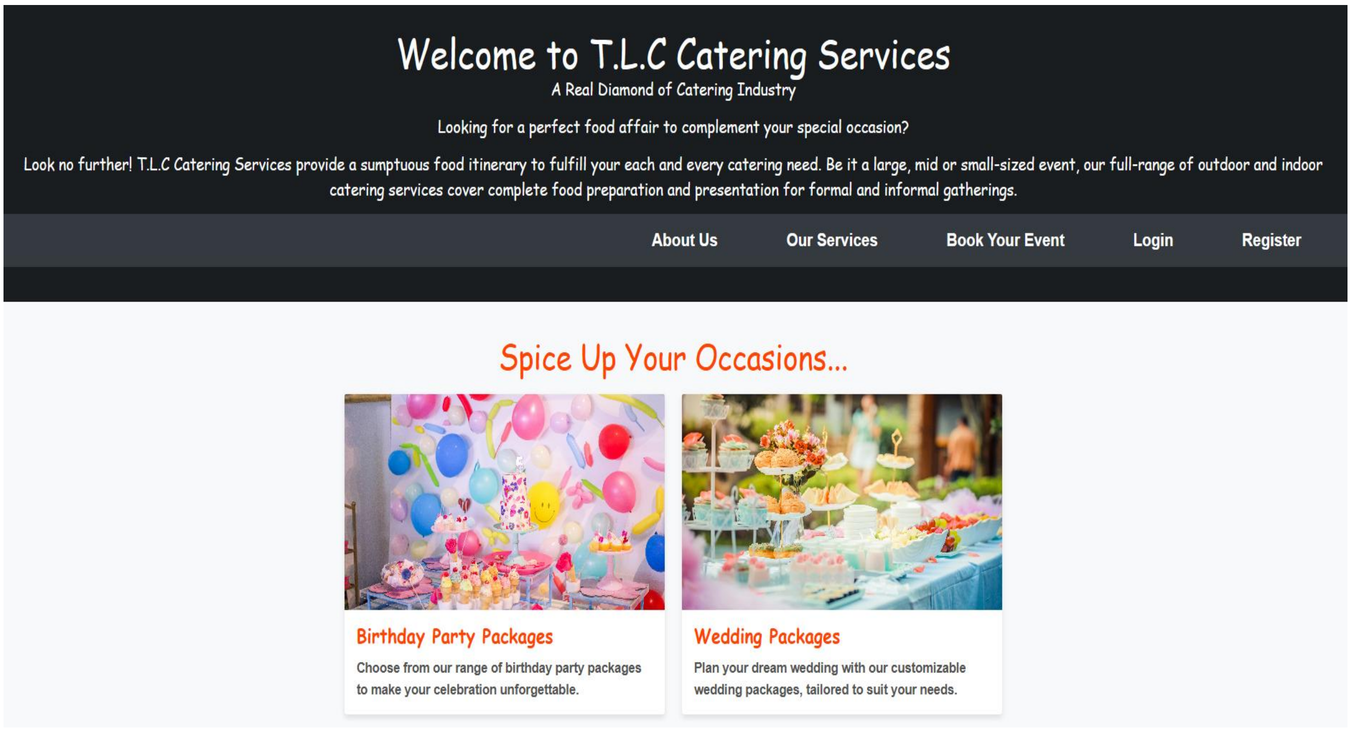
Field Name	Data Type	Attributes	Description
id	long	Primary key	Unique identifier for each package
name	nvarchar(50)	not null	Name/title of the package
description	nvarchar(200)	not null	Detailed description of what the package offers
type	nvarchar(50)	not null	Category of the event (e.g., wedding, birthday)
image	nvarchar(50)	not null	Filename or path to image representing package
max_attendees	int	Not null	Maximum number of guests that can attend

Table 4: bookings – fields and data types

Field Name	Data Type	Attributes	Description
id	long	Primary key	Unique identifier for each booking
event_type	nvarchar(50)	not null	Type of event being booked (e.g., wedding, birthday)
event_date	date	not null	Scheduled date of the event
event_time	time(7)	not null	Scheduled time of the event
attendees	int	not null	Number of guests attending the event
location	nvarchar(50)	not null	Address or venue of the event
userID	int	Foreign key, not null	Reference to the user who made the booking
packageID	int	Foreign key, not null	Reference to the selected catering package

2.4GUI Project

Home Page:



Welcome to T.L.C Catering Services

A Real Diamond of Catering Industry

Looking for a perfect food affair to complement your special occasion?

Look no further! T.L.C Catering Services provide a sumptuous food itinerary to fulfill your each and every catering need. Be it a large, mid or small-sized event, our full-range of outdoor and indoor catering services cover complete food preparation and presentation for formal and informal gatherings.

About Us | Our Services | Book Your Event | Login | Register

Spice Up Your Occasions...

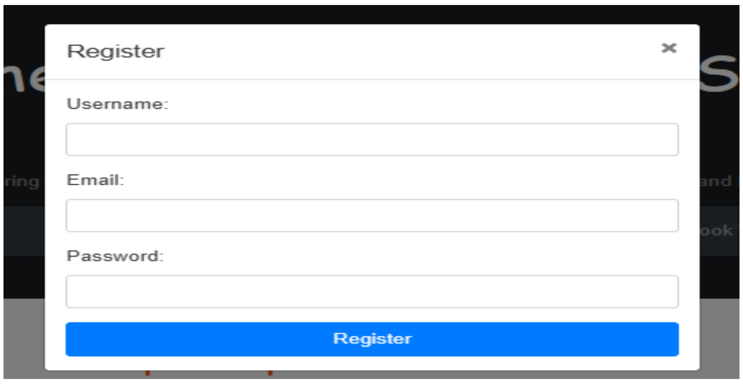
Birthday Party Packages

Choose from our range of birthday party packages to make your celebration unforgettable.

Wedding Packages

Plan your dream wedding with our customizable wedding packages, tailored to suit your needs.

Registration and Login:



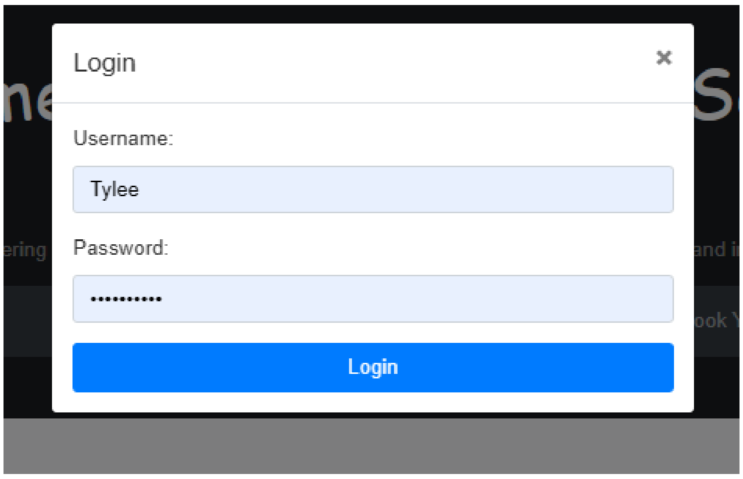
Register

Username:

Email:

Password:

Register



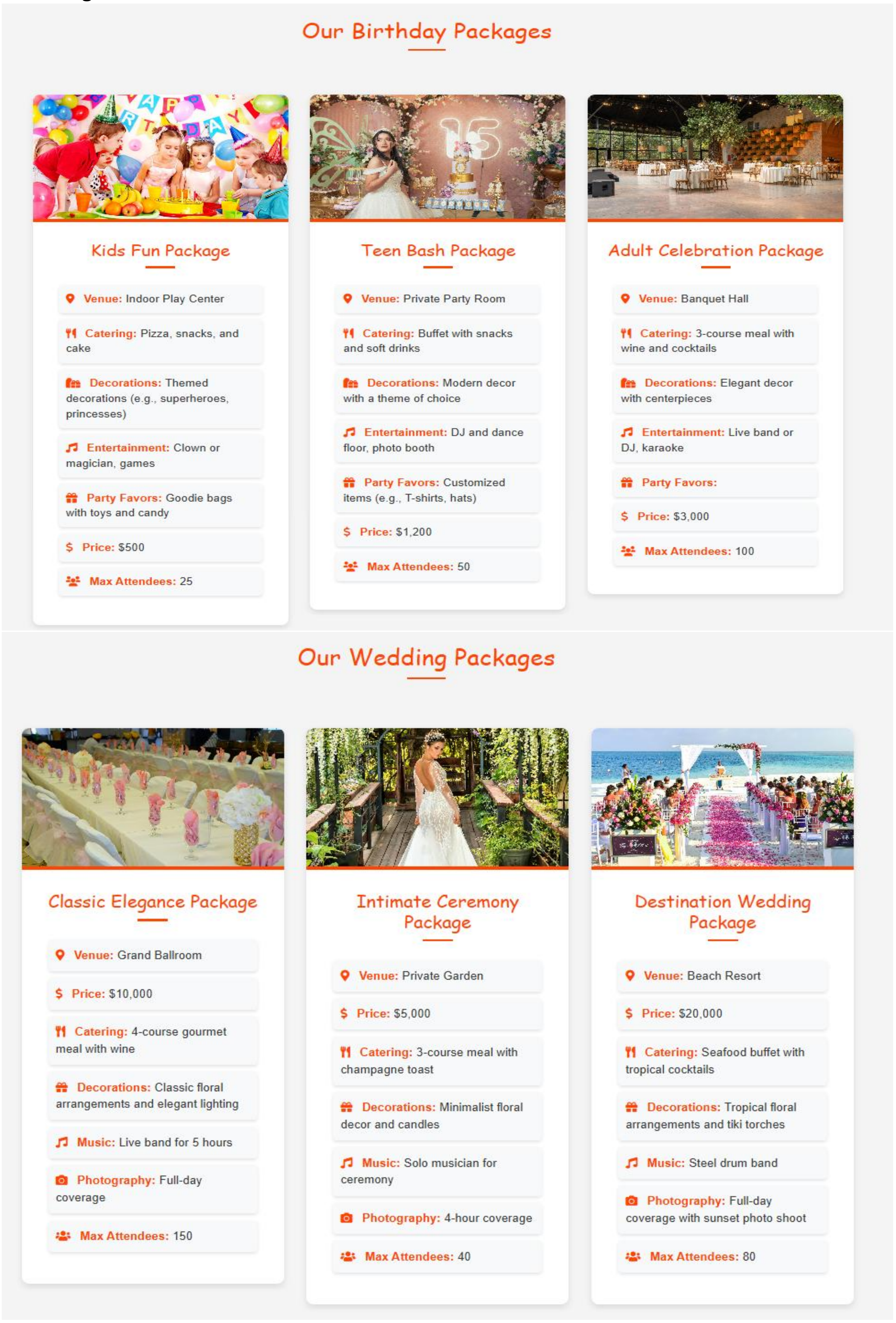
Login

Username: Tylee

Password: ••••••••••

Login

Packages:



Our Birthday Packages

Kids Fun Package

- **Venue:** Indoor Play Center
- **Catering:** Pizza, snacks, and cake
- **Decorations:** Themed decorations (e.g., superheroes, princesses)
- **Entertainment:** Clown or magician, games
- **Party Favors:** Goodie bags with toys and candy
- **Price:** $500
- **Max Attendees:** 25

Teen Bash Package

- **Venue:** Private Party Room
- **Catering:** Buffet with snacks and soft drinks
- **Decorations:** Modern decor with a theme of choice
- **Entertainment:** DJ and dance floor, photo booth
- **Party Favors:** Customized items (e.g., T-shirts, hats)
- **Price:** $1,200
- **Max Attendees:** 50

Adult Celebration Package

- **Venue:** Banquet Hall
- **Catering:** 3-course meal with wine and cocktails
- **Decorations:** Elegant decor with centerpieces
- **Entertainment:** Live band or DJ, karaoke
- **Party Favors:**
- **Price:** $3,000
- **Max Attendees:** 100

Our Wedding Packages

Classic Elegance Package

- **Venue:** Grand Ballroom
- **Price:** $10,000
- **Catering:** 4-course gourmet meal with wine
- **Decorations:** Classic floral arrangements and elegant lighting
- **Music:** Live band for 5 hours
- **Photography:** Full-day coverage
- **Max Attendees:** 150

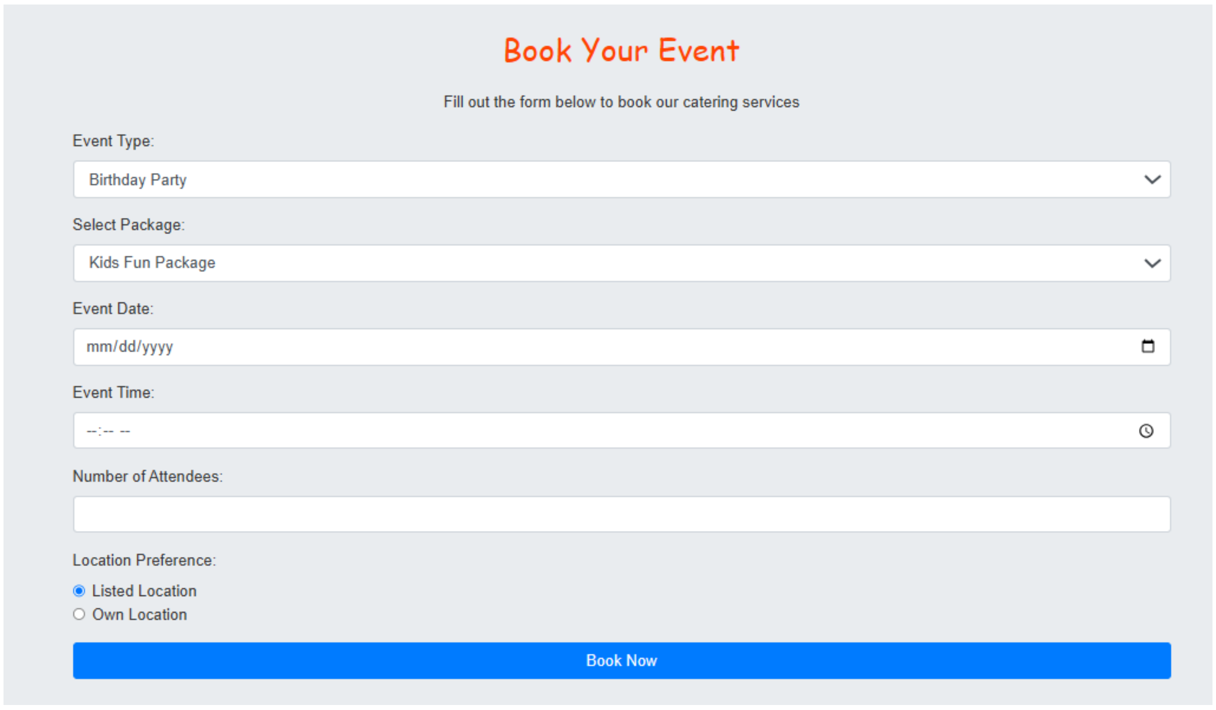
Intimate Ceremony Package

- **Venue:** Private Garden
- **Price:** $5,000
- **Catering:** 3-course meal with champagne toast
- **Decorations:** Minimalist floral decor and candles
- **Music:** Solo musician for ceremony
- **Photography:** 4-hour coverage
- **Max Attendees:** 40

Destination Wedding Package

- **Venue:** Beach Resort
- **Price:** $20,000
- **Catering:** Seafood buffet with tropical cocktails
- **Decorations:** Tropical floral arrangements and tiki torches
- **Music:** Steel drum band
- **Photography:** Full-day coverage with sunset photo shoot
- **Max Attendees:** 80

Booking:



Book Your Event

Fill out the form below to book our catering services

Event Type: Birthday Party

Select Package: Kids Fun Package

Event Date: mm/dd/yyyy

Event Time: --:-- --

Number of Attendees:

Location Preference:

● Listed Location
○ Own Location

3. DEVELOPMENT OF THE CATERING SERVICES BOOKING APPLICATION

Introduction

This chapter outlines the practical implementation of the Catering Services Booking Application. It discusses the selected tools and technologies, the development environment setup, implementation details, and testing procedures used to ensure system reliability and usability.

3.1 Choice of Tools and Technology

The system was developed using a collection of open-source technologies that prioritize simplicity, maintainability, and scalability. Each tool was selected based on its specific strengths and seamless integration with the overall architecture.

Flask

Flask is a micro web framework for Python that provides routing, request handling, and templating capabilities. Its minimalist design allowed for flexible architecture decisions without enforcing unnecessary constraints. Flask was responsible for:

- Handling HTTP requests and form submissions.
- Managing routing logic (@app.route).
- Integrating with session management and authentication modules.
- Coordinating responses between the front end and the database layer.

SQLite

SQLite was used as the relational database engine during development due to its ease of setup and integration with Flask. Its self-contained architecture eliminated the need for server configuration, making it suitable for lightweight applications.

- Tables were defined using SQLAlchemy ORM to manage models for Users, Bookings, Packages, and Inquiries.
- Foreign key constraints ensured data consistency between relational entities.

Flask-Login

Flask-Login was integrated to handle user authentication and session management. It ensured that protected routes were only accessible by logged-in users and differentiated between regular users and administrators. Sessions persisted until logout, providing a secure and intuitive experience.

Werkzeug

Werkzeug provided essential security utilities such as password hashing and verification using generate_password_hash() and check_password_hash(). These measures ensured that plaintext passwords were never stored, aligning with best security practices.

Jinja2

Jinja2 is Flask’s default templating engine. It was used to dynamically inject Python variables into HTML pages and render user-specific content such as personalized dashboards, booking lists, and admin panels.

Bootstrap

Bootstrap 5 was used to implement a responsive, mobile-first interface. Its predefined grid system and components reduced design overhead and enabled consistent UI styling across all pages, including:

- Navigation bar
- Cards for package listings
- Modal forms and alerts
- Responsive form layouts

JavaScript and AJAX

JavaScript provided interactivity on the client side, such as form validation and dynamic content updates. AJAX (Asynchronous JavaScript and XML) was used for submitting forms and retrieving filtered data without requiring full page reloads. This improved the responsiveness and perceived performance of the application.

HTML and CSS

HTML structured the web content, while CSS was used for custom styling beyond Bootstrap defaults. Layout, typography, and button behavior were fine-tuned to maintain a professional and user-friendly design.

Python

Python served as the primary language for the backend, enabling quick development of logic and database interactions. Its simplicity and extensive library support made it ideal for integrating web, database, and security components.

3.2 Development Environment

The development environment was configured to ensure a balance between efficiency and portability. The tools and configurations below supported the entire lifecycle from code writing and debugging to testing and version control.

Component	Configuration
Programming Language	Python 3.11
IDE	Visual Studio Code
Web Framework	Flask
Database	SQLite
Templating Engine	Jinja2
Front-End Technologies	HTML5, CSS3, JavaScript, Bootstrap 5
Testing Tools	Postman, browser developer tools
Version Control System	Git with GitHub repository
Package Management	pip (Python libraries)

The application was run and tested in a local development server using the flask run command. Git was used for version tracking and collaboration.

3.3 System Detailed Design

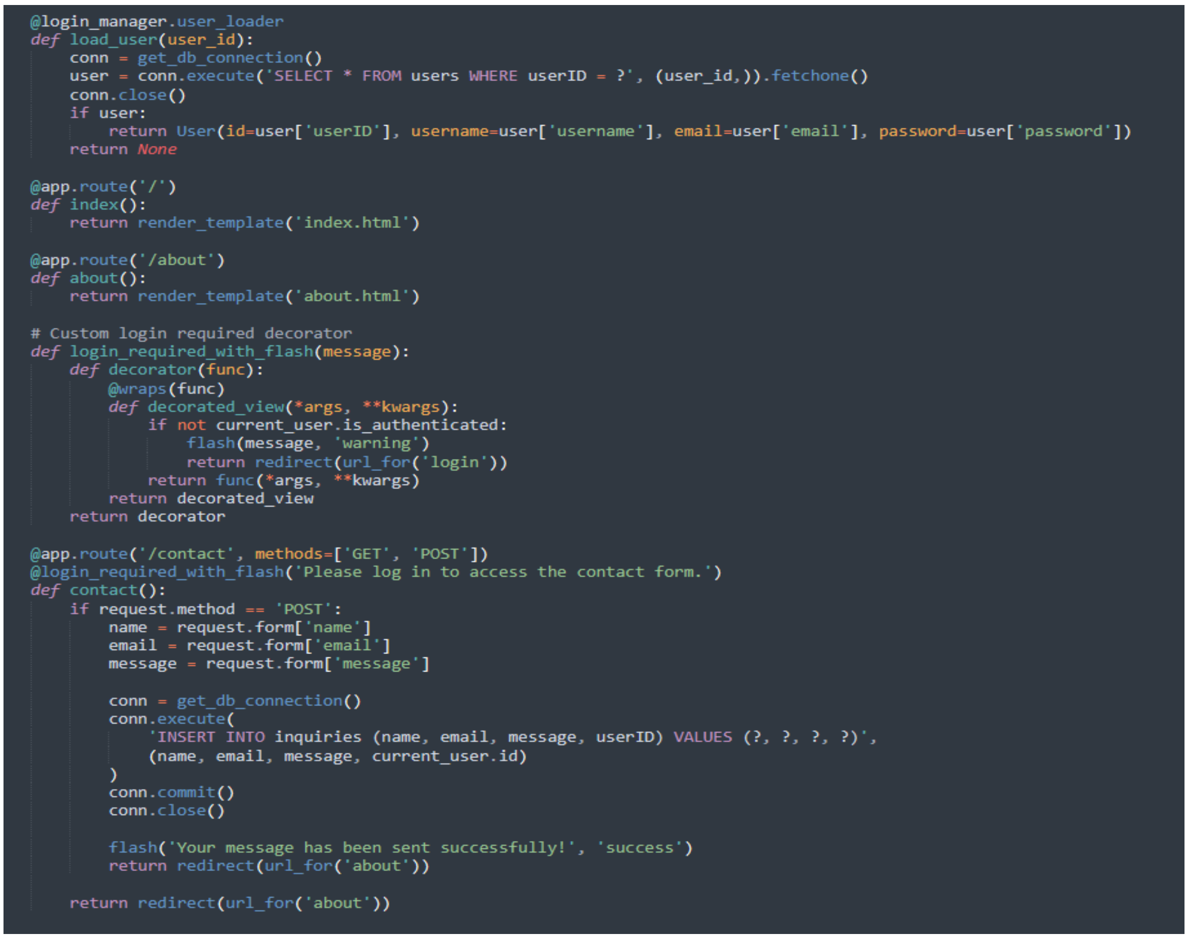
The Catering Services Booking Application was built using the Model-View-Controller (MVC) design pattern. The architecture separates the business logic (controllers), data layer (models/database), and presentation layer (views) to ensure maintainability and scalability.

3.3.1 Backend Logic

The backend was developed using Flask, with clear routing logic for each major feature:

User Authentication

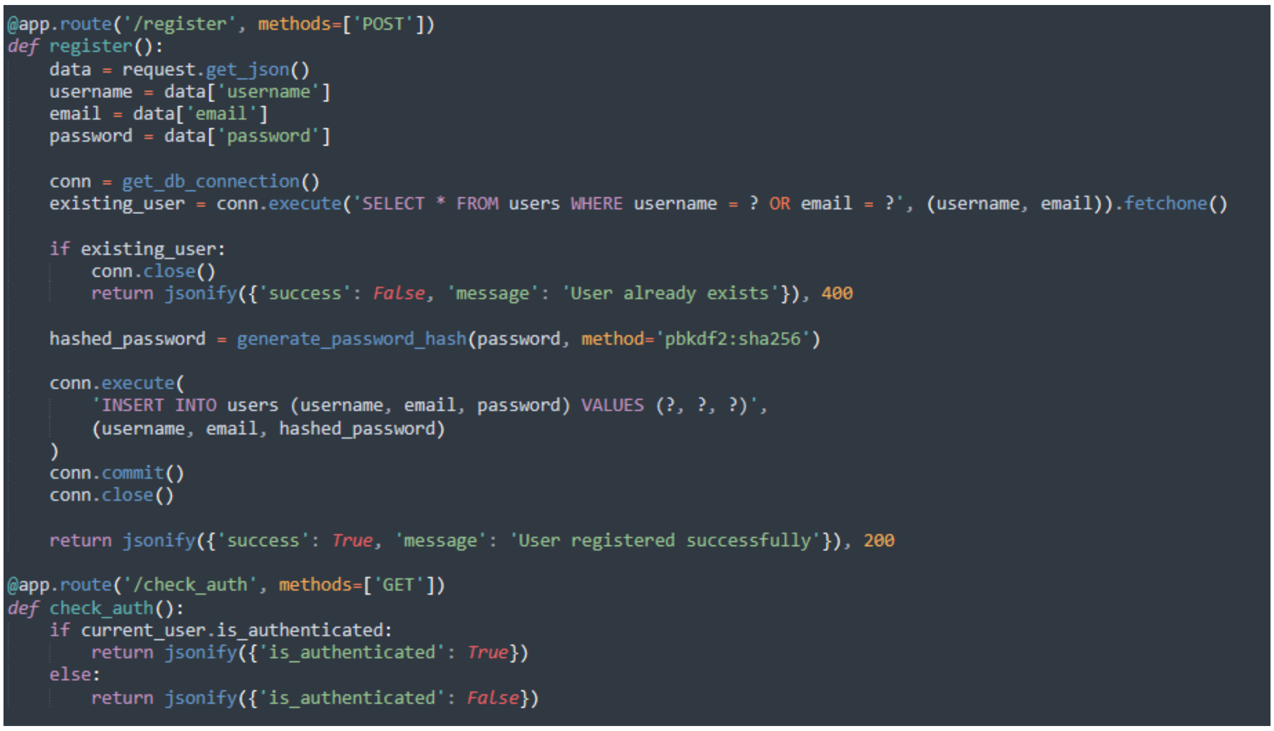
Users register and log in securely via hashed passwords using Werkzeug. Flask-Login manages user sessions, ensuring persistent login and automatic logout handling. Access to protected routes is enforced via the @login_required decorator or a custom version @login_required_with_flash for personalized feedback.



```python
@login_manager.user_loader
def load_user(user_id):
    conn = get_db_connection()
    user = conn.execute('SELECT * FROM users WHERE userID = ?', (user_id,)).fetchone()
    conn.close()
    if user:
        return User(id=user['userID'], username=user['username'], email=user['email'], password=user['password'])
    return None

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/about')
def about():
    return render_template('about.html')

# Custom login required decorator
def login_required_with_flash(message):
    def decorator(func):
        @wraps(func)
        def decorated_view(*args, **kwargs):
            if not current_user.is_authenticated:
                flash(message, 'warning')
                return redirect(url_for('login'))
            return func(*args, **kwargs)
        return decorated_view
    return decorator

@app.route('/contact', methods=['GET', 'POST'])
@login_required_with_flash('Please log in to access the contact form.')
def contact():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        message = request.form['message']

        conn = get_db_connection()
        conn.execute(
            'INSERT INTO inquiries (name, email, message, userID) VALUES (?, ?, ?, ?)',
            (name, email, message, current_user.id)
        )
        conn.commit()
        conn.close()

        flash('Your message has been sent successfully!', 'success')
        return redirect(url_for('about'))

    return redirect(url_for('about'))
```

Registration Route

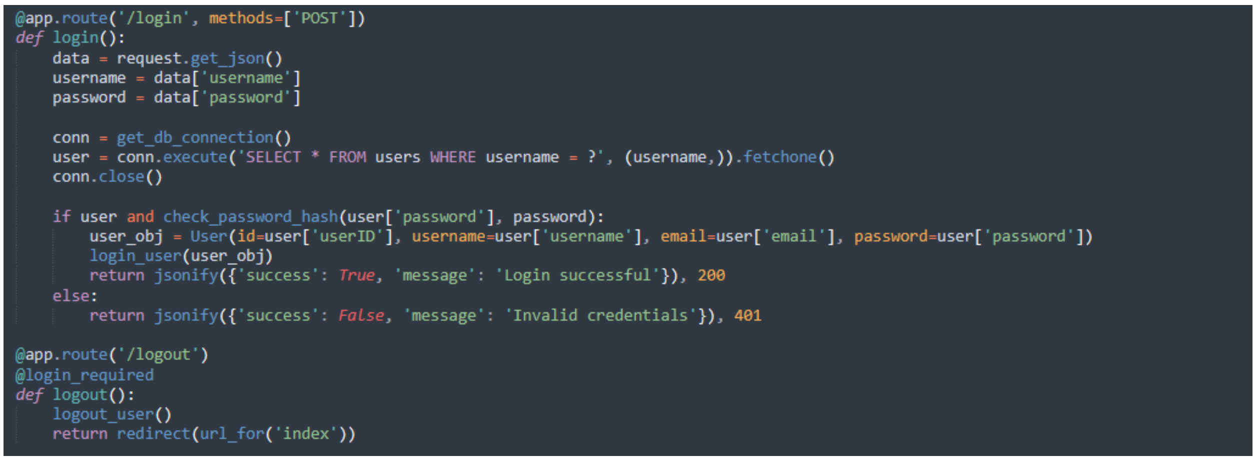
Accepts JSON input, checks for existing users, hashes the password, and inserts a new user.



```python
@app.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    username = data['username']
    email = data['email']
    password = data['password']

    conn = get_db_connection()
    existing_user = conn.execute('SELECT * FROM users WHERE username = ? OR email = ?', (username, email)).fetchone()

    if existing_user:
        conn.close()
        return jsonify({'success': False, 'message': 'User already exists'}), 400

    hashed_password = generate_password_hash(password, method='pbkdf2:sha256')

    conn.execute(
        'INSERT INTO users (username, email, password) VALUES (?, ?, ?)',
        (username, email, hashed_password)
    )
    conn.commit()
    conn.close()

    return jsonify({'success': True, 'message': 'User registered successfully'}), 200

@app.route('/check_auth', methods=['GET'])
def check_auth():
    if current_user.is_authenticated:
        return jsonify({'is_authenticated': True})
    else:
        return jsonify({'is_authenticated': False})
```

Login Route

Verifies credentials and initiates a session. For logout, session is terminated and user is redirected:

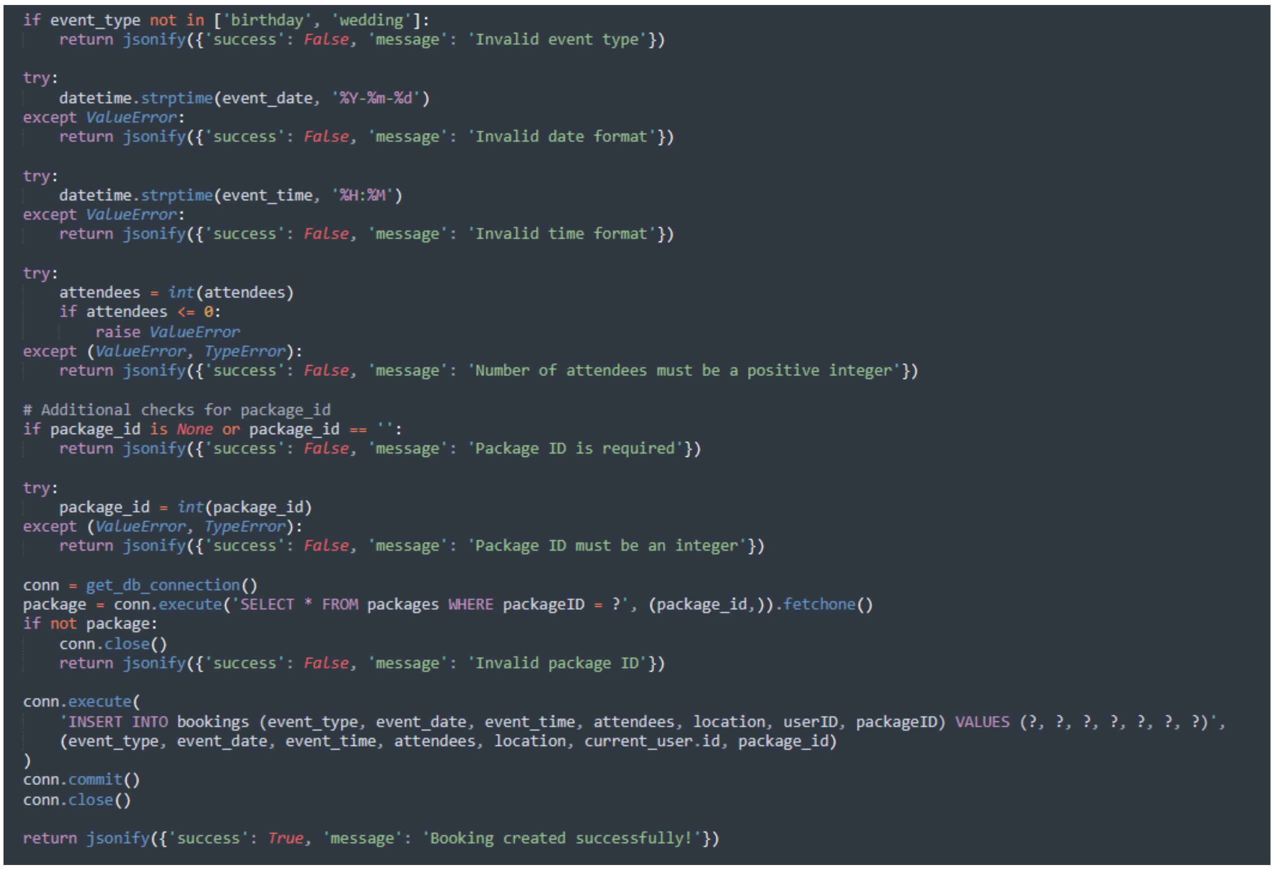


```python
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    username = data['username']
    password = data['password']

    conn = get_db_connection()
    user = conn.execute('SELECT * FROM users WHERE username = ?', (username,)).fetchone()
    conn.close()

    if user and check_password_hash(user['password'], password):
        user_obj = User(id=user['userID'], username=user['username'], email=user['email'], password=user['password'])
        login_user(user_obj)
        return jsonify({'success': True, 'message': 'Login successful'}), 200
    else:
        return jsonify({'success': False, 'message': 'Invalid credentials'}), 401

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('index'))
```

3.3.2 Booking Logic and Validation

The booking system is designed to reject invalid data through layered validation checks:

- Event type must be either 'wedding' or 'birthday'
- Dates and times are parsed and validated using datetime.strptime
- Guest count must be a positive integer
- Package ID must exist in the database

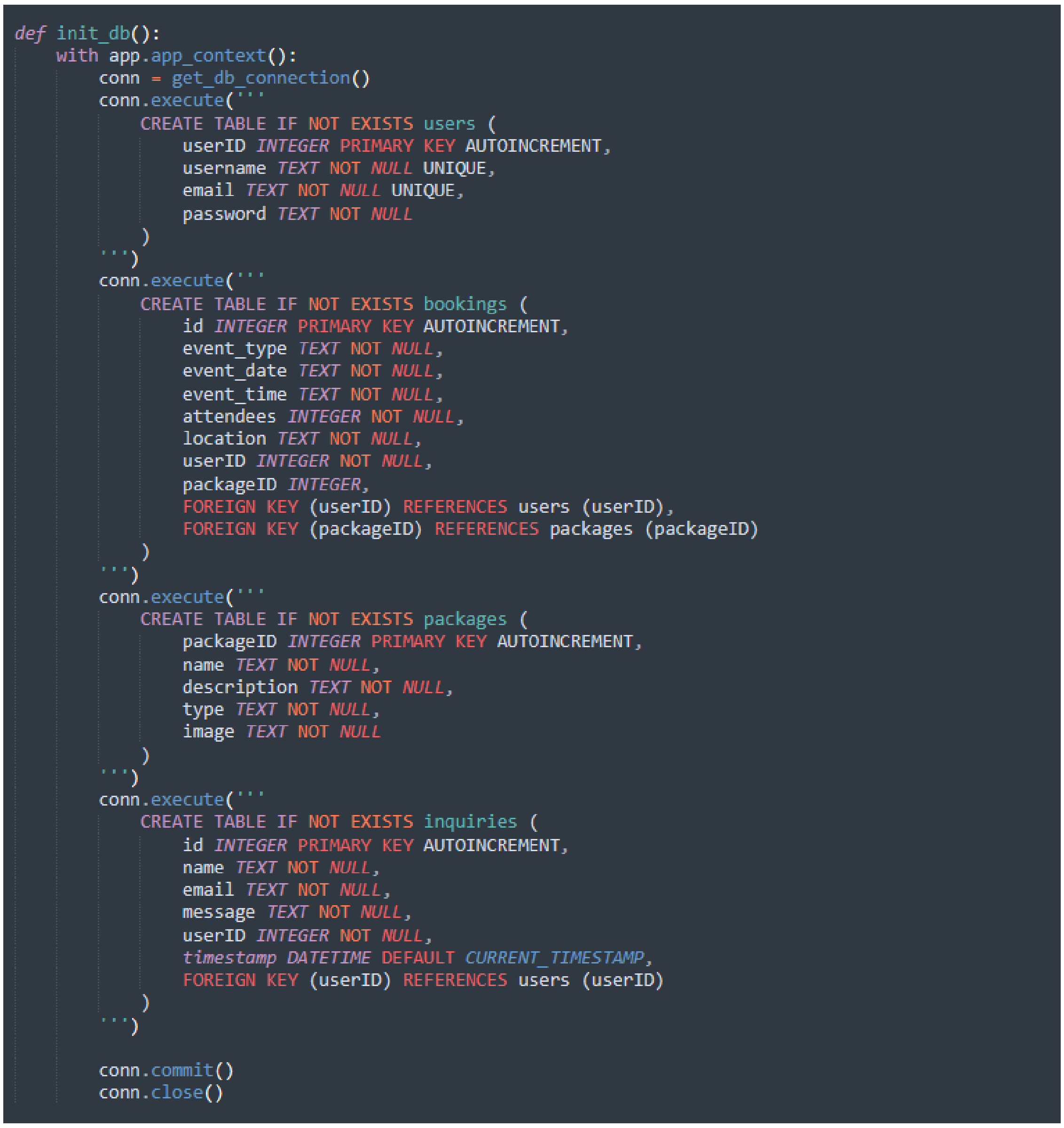


```python
if event_type not in ['birthday', 'wedding']:
    return jsonify({'success': False, 'message': 'Invalid event type'})

try:
    datetime.strptime(event_date, '%Y-%m-%d')
except ValueError:
    return jsonify({'success': False, 'message': 'Invalid date format'})

try:
    datetime.strptime(event_time, '%H:%M')
except ValueError:
    return jsonify({'success': False, 'message': 'Invalid time format'})

try:
    attendees = int(attendees)
    if attendees <= 0:
        raise ValueError
except (ValueError, TypeError):
    return jsonify({'success': False, 'message': 'Number of attendees must be a positive integer'})

# Additional checks for package_id
if package_id is None or package_id == '':
    return jsonify({'success': False, 'message': 'Package ID is required'})

try:
    package_id = int(package_id)
except (ValueError, TypeError):
    return jsonify({'success': False, 'message': 'Package ID must be an integer'})

conn = get_db_connection()
package = conn.execute('SELECT * FROM packages WHERE packageID = ?', (package_id,)).fetchone()
if not package:
    conn.close()
    return jsonify({'success': False, 'message': 'Invalid package ID'})

conn.execute(
    'INSERT INTO bookings (event_type, event_date, event_time, attendees, location, userID, packageID) VALUES (?, ?, ?, ?, ?, ?, ?)',
    (event_type, event_date, event_time, attendees, location, current_user.id, package_id)
)
conn.commit()
conn.close()

return jsonify({'success': True, 'message': 'Booking created successfully!'})
```

3.3.3 Database Initialization and Seeding

The system includes an initialization routine that creates the necessary tables on first run.

Additionally, default packages are automatically inserted if none exist, using a bulk insert script.



```python
def init_db():
    with app.app_context():
        conn = get_db_connection()
        conn.execute('''
            CREATE TABLE IF NOT EXISTS users (
                userID INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT NOT NULL UNIQUE,
                email TEXT NOT NULL UNIQUE,
                password TEXT NOT NULL
            )
        ''')
        conn.execute('''
            CREATE TABLE IF NOT EXISTS bookings (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                event_type TEXT NOT NULL,
                event_date TEXT NOT NULL,
                event_time TEXT NOT NULL,
                attendees INTEGER NOT NULL,
                location TEXT NOT NULL,
                userID INTEGER NOT NULL,
                packageID INTEGER,
                FOREIGN KEY (userID) REFERENCES users (userID),
                FOREIGN KEY (packageID) REFERENCES packages (packageID)
            )
        ''')
        conn.execute('''
            CREATE TABLE IF NOT EXISTS packages (
                packageID INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                description TEXT NOT NULL,
                type TEXT NOT NULL,
                image TEXT NOT NULL
            )
        ''')
        conn.execute('''
            CREATE TABLE IF NOT EXISTS inquiries (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                email TEXT NOT NULL,
                message TEXT NOT NULL,
                userID INTEGER NOT NULL,
                timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
                FOREIGN KEY (userID) REFERENCES users (userID)
            )
        ''')

        conn.commit()
        conn.close()
```

3.4 Tests

To ensure the reliability, functionality, and usability of the Catering Services Booking Application, a series of manual and automated tests were performed. Testing focused on core user actions such as registration, login, event booking, and inquiry submission. Each test scenario included input validation, expected system response, and verification of backend changes (e.g., database updates).

Test Scenarios Overview

Test Case ID	Test Scenario	Test Steps	Expected Outcome	Actual Outcome	Status
TC001	User Registration	1. Navigate to registration page 2. Enter valid username, email, password 3. Click "Register"	User is registered and redirected to login page	User registration successful	Passed
TC002	User Login	1. Navigate to login page 2. Enter valid credentials 3. Click "Login"	User is logged in and redirected to homepage	User successfully logged in	Passed
TC003	Services Browsing	1. Navigate to "Services" 2. Select a package category 3. Browse options	Package categories and options are displayed	As expected	Passed
TC004	Contact Submission	1. Go to "Contact" 2. Enter name, email, and message 3. Click "Send"	Inquiry is submitted and saved to the database	As expected	Passed
TC005	Event Booking	1. Go to "Book Your Event" 2. Fill in details 3. Click "Book Now"	Booking confirmation displayed and data stored in DB	As expected	Passed
TC006	Logout	1. Navigate to "Logout" 2. Click on "Logout"	User is logged out and redirected to login or home page	As expected	Passed
TC007	Register with existing username/email	1. Go to registration page 2. Enter an existing username/email 3. Click "Register"	User sees error: "Username/email already exists"	System allows duplicate registration	Failed
TC008	Submit booking form with missing fields	1. Open booking form 2. Leave required fields empty 3. Click "Book Now"	Form shows validation errors and is not submitted	Incomplete form submitted, data may be lost or inconsistent	Failed

3.4.1 Test Scenario – Event Booking (TC005)

Objective
To verify that a logged-in user can successfully book a catering service by completing the event booking form and that the system processes and stores the data correctly.

Preconditions

- The user is logged in to their account.
- At least one catering package exists in the system (e.g., "Kids Fun Package").

Test Steps

1. Navigate to the "Book Your Event" page.
2. Fill in the booking form with the following:
    - **Event Type**: Birthday Party
    - **Package**: Kids Fun Package
    - **Event Date**: 2025-05-24
    - **Event Time**: 11:00 AM
    - **Number of Attendees**: 60
    - **Location Preference**: Listed Location
3. Click the “Book Now” button.

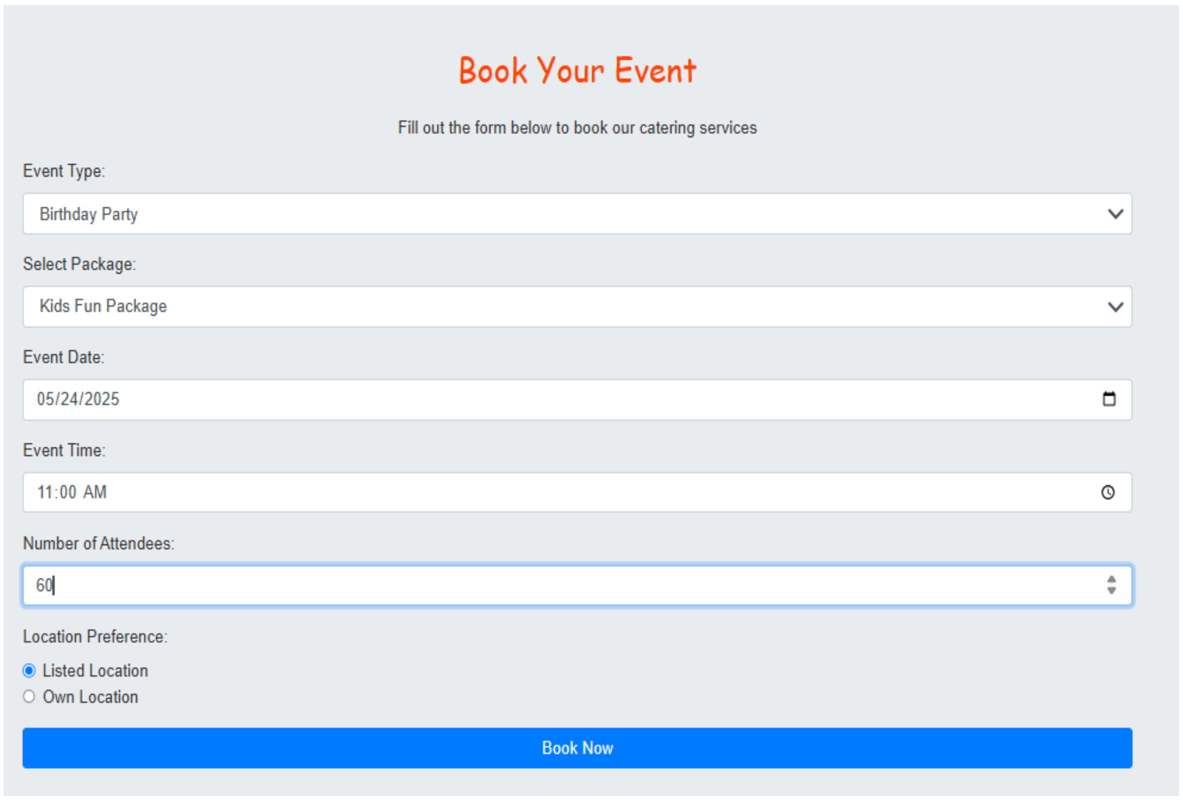
Expected Outcome

- A success message is displayed (*"Booking successful!"*).
- Booking is saved in the database with accurate details.
- Booking appears in the user’s booking history or admin dashboard.

Actual Outcome

- The form was submitted successfully.
- Booking confirmation appeared.
- Data was stored correctly in the database and confirmed.

Status: **Passed**



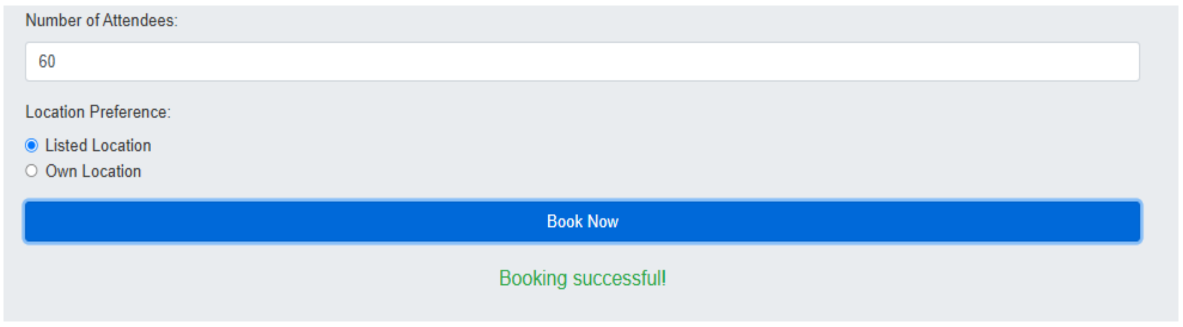
Book Your Event

Fill out the form below to book our catering services

Event Type:	Birthday Party
Select Package:	Kids Fun Package
Event Date:	05/24/2025
Event Time:	11:00 AM
Number of Attendees:	60

Location Preference:

[✓ Listed Location] Own Location



Number of Attendees:	60

Location Preference:

[✓ Listed Location] Own Location

Booking successful!

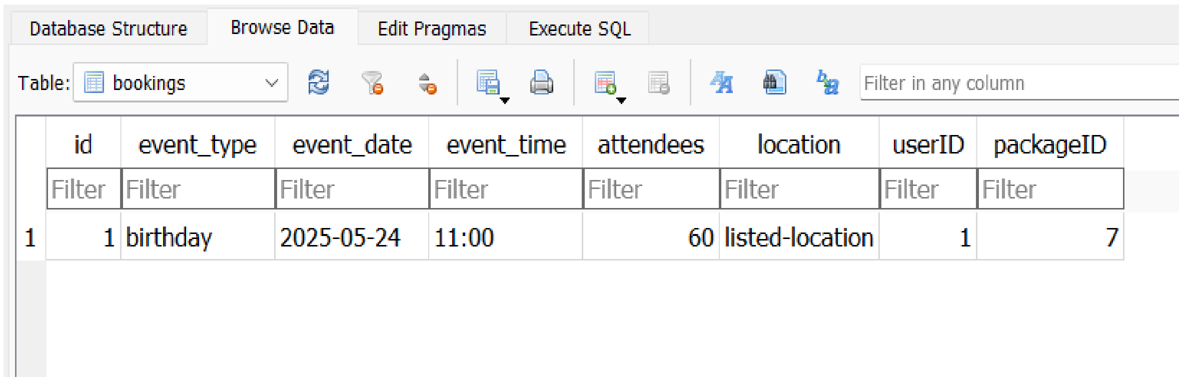


Table: bookings

	id	event_type	event_date	event_time	attendees	location	userID	packageID
1	1	birthday	2025-05-24	11:00	60	listed-location	1	7

4. PRACTICAL USE CASE

This section demonstrates how a real user would interact with the Catering Services Booking Application to complete a full booking process. Screenshots are included to illustrate each key step.

Scenario

User: John Smith

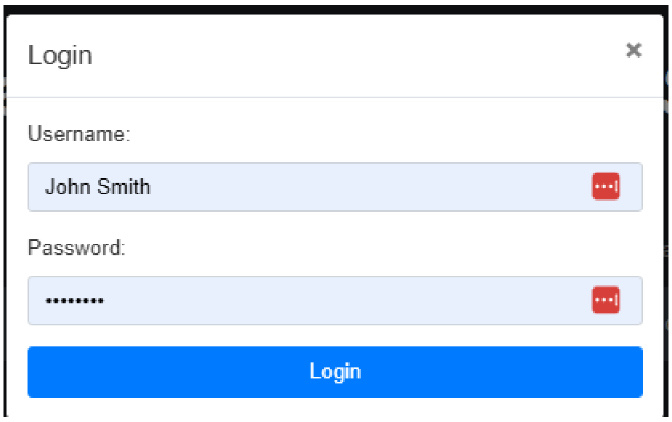
Goal: Book catering services for his son's birthday party using the online booking system.

Step-by-Step Walkthrough

Step 1: User Logs In

John visits the application and logs in using his registered email and password.

<u>Login form with filled-in credentials and login button.</u>


Login

Username:

John Smith

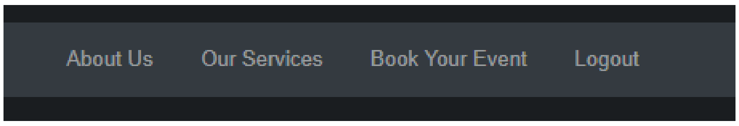
Password:

••••••••

Step 2: Navigate to the Booking Page

After logging in, John clicks on the **"Book Your Event"** tab in the main menu.

<u>Dashboard or navigation bar with the "Book Your Event" link.</u>



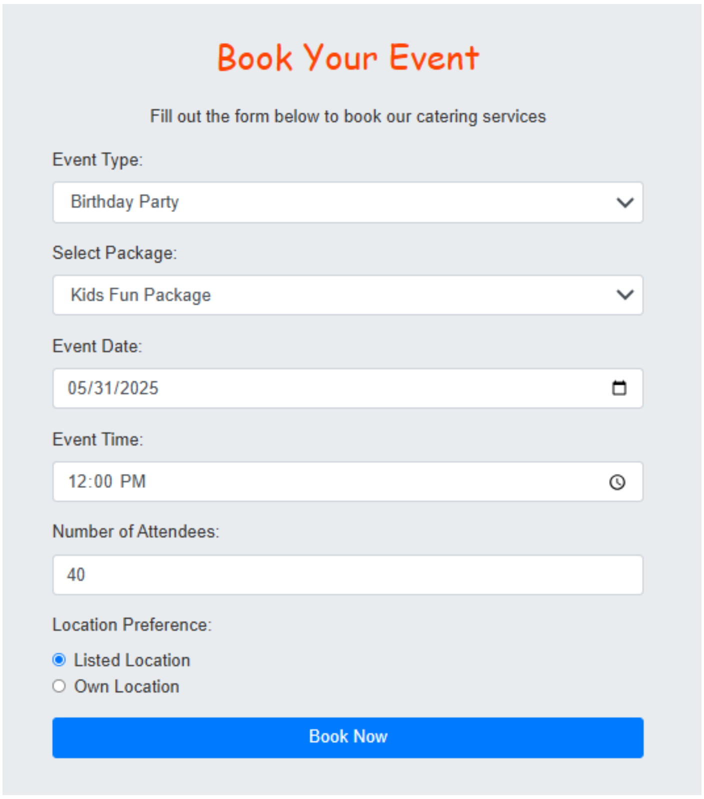
Step 3: Fill Out the Booking Form

On the booking page, John fills out the form with the following details:

- **Event Type:** Birthday Party
- **Select Package:** Kids Fun Package
- **Event Date:** 31/05/2025
- **Event Time:** 12:00 PM
- **Number of Attendees:** 40
- **Location Preference:** Listed Location

He then clicks the **"Book Now"** button.

<u>Booking form before submission</u>



Book Your Event

Fill out the form below to book our catering services

Event Type:
Birthday Party

Select Package:
Kids Fun Package

Event Date:
05/31/2025

Event Time:
12:00 PM

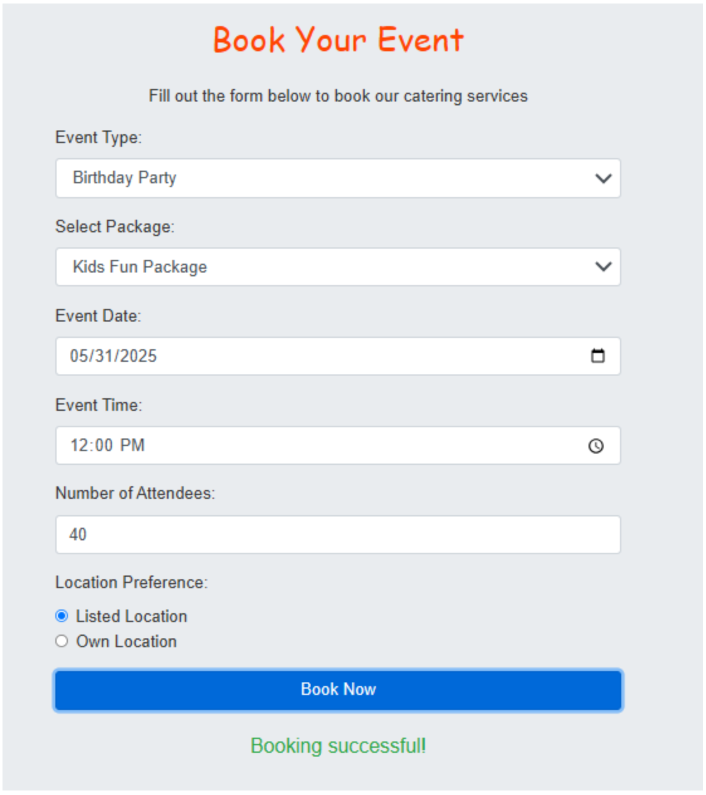
Number of Attendees:
40

Location Preference:
● Listed Location
○ Own Location

Step 4: Booking Confirmation

The system processes the request and displays a success message confirming that the booking was successful.

<u>Success message after booking submission</u>



Book Your Event

Fill out the form below to book our catering services

Event Type:	Birthday Party
Select Package:	Kids Fun Package
Event Date:	05/31/2025
Event Time:	12:00 PM
Number of Attendees:	40
Location Preference:	[✓ Listed Location] Own Location

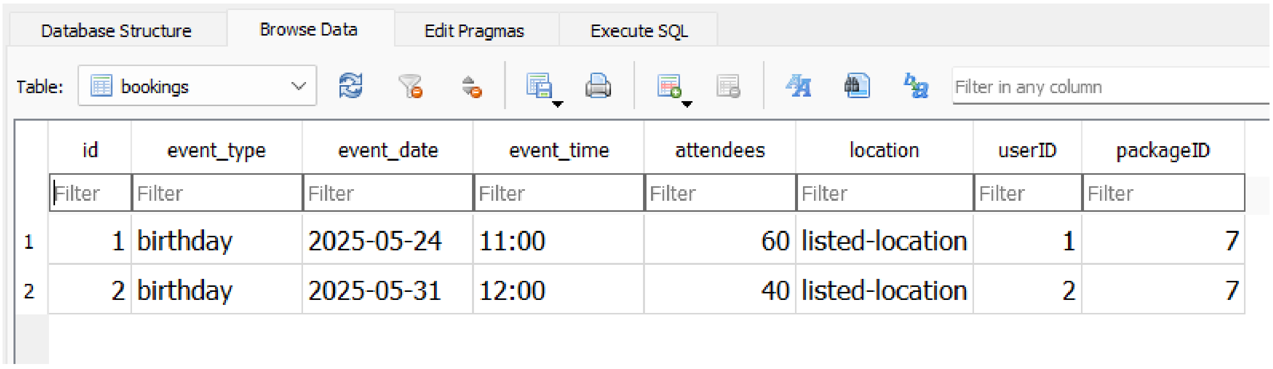
Book Now

Booking successful!

Step 5: Booking Stored in System

The booking is saved in the backend database. It becomes visible in the user's booking history and is also accessible to the admin for preparation and follow-up.

<u>Database entry or admin panel showing saved booking</u>



	id	event_type	event_date	event_time	attendees	location	userID	packageID
1	1	birthday	2025-05-24	11:00	60	listed-location	1	7
2	2	birthday	2025-05-31	12:00	40	listed-location	2	7

Result

John successfully completes the event booking process in under 5 minutes. The system handled input validation, stored data securely, and provided immediate user feedback demonstrating the efficiency and user-friendliness of the platform.

5. SUMMARY

This thesis researched and developed a web-based **Catering Services Booking Application** aimed at improving how clients book catering services for events such as birthday parties and weddings. The system replaces traditional, manual methods with a digital solution that is more efficient, user-friendly, and accessible.

The application was built using **Flask** for the backend, **SQLite** for data storage, and **HTML, CSS, JavaScript**, and **Bootstrap** for the front end. Key features include user authentication, event booking, package browsing, and an admin panel for managing inquiries and services.

Testing confirmed that the application performs reliably, handles user input correctly, and stores booking data securely. A practical use case demonstrated how real users can interact with the system to complete bookings quickly and easily.

The project highlights how web technology can be effectively used to automate service-based processes. Future enhancements could include features such as online payments, email confirmations, and a more advanced admin dashboard.

6. BIBLIOGRAPHY

Books and Publications

1. Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python* (2nd ed.). O'Reilly Media.
2. Sommerville, I. (2015). *Software Engineering* (10th ed.). Pearson Education.
3. Welling, L., & Thomson, L. (2009). *PHP and MySQL Web Development* (4th ed.). Addison-Wesley.
4. Pressman, R. S. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education.

Online resources

5. Flask Documentation. (n.d.). Retrieved from: https://flask.palletsprojects.com/
6. SQLite Documentation. (n.d.). Retrieved from: https://www.sqlite.org/docs.html
7. Bootstrap 5 Documentation. (n.d.). Retrieved from: https://getbootstrap.com/docs/5.0/getting-started/introduction/
8. Flask-Login Documentation. (n.d.). Retrieved from: https://flask-login.readthedocs.io/en/latest/
9. Werkzeug Security. (n.d.). Retrieved from: https://werkzeug.palletsprojects.com/
10. GitHub Docs. (n.d.). Retrieved from: https://docs.github.com/