Web App Design 3

Case of a Hotel Booking in Frontend

National Taipei University of Technology / Eric Xu

Objective

Frontend web site

Database design

Integration

Frontend Web Site

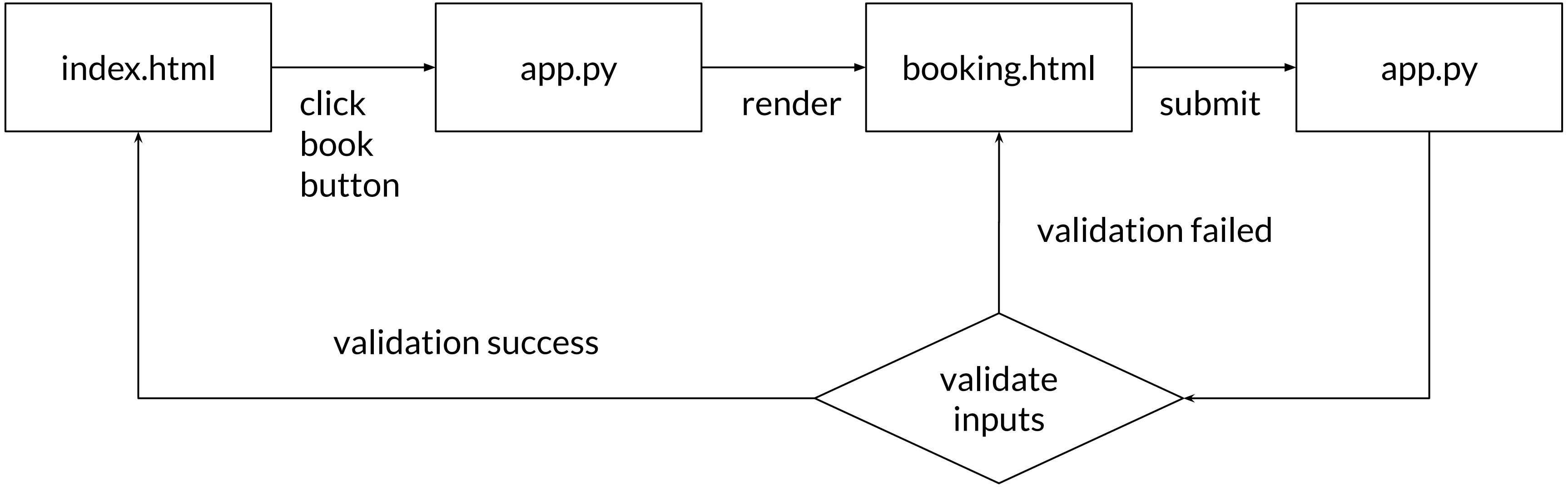
Project Structure

```
hotel_booking/
│
├── templates/
│   ├── index.html
│   └── booking.html
│
├── requirements.txt
└── app.py
```

```
conda create -n hotel python=3.11
conda activate hotel
pip install -r requirements.txt
```

Flask
Flask-WTF
WTForms

Processing Flow



app.py

```
from flask import Flask, render_template, request, redirect,
url_for
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField, DateField,
SelectField
from wtforms.validators import DataRequired

app = Flask(__name__)
app.config['SECRET_KEY'] = 'YourSecretKey'

# Example room data (in a real application, this would come from a
database)
rooms = [
    ('101', 'Single Room'),
    ('102', 'Double Room'),
    ('103', 'Deluxe Room')
]

class BookingForm(FlaskForm):
    guest_name = StringField('Guest Name',
validators=[DataRequired()])
    room_number = SelectField('Room Number', choices=rooms,
validators=[DataRequired()])
    check_in_date = DateField('Check-In Date', format='%Y-%m-%d',
validators=[DataRequired()])
check_out_date = DateField('Check-Out Date',
format='%Y-%m-%d', validators=[DataRequired()])
    contact_info = StringField('Contact Information',
validators=[DataRequired()])
    submit = SubmitField('Book Now')

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

@app.route('/booking', methods=['GET', 'POST'])
def booking():
    form = BookingForm()
    if form.validate_on_submit():
        for field in form:
            print(f"{field.name}: {field.data}")
        return redirect(url_for('index'))
    return render_template('booking.html',  form=form)

if __name__ == '__main__':
    app.run(debug=True)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Hotel Booking</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootst
rap.min.css">
</head>
<body>
    <div class="container">
        <h1>Welcome to Our Hotel</h1>
        <a href="{{ url_for('booking') }}" class="btn
btn-primary">Book a Room</a>
    </div>
</body>
</html>
```

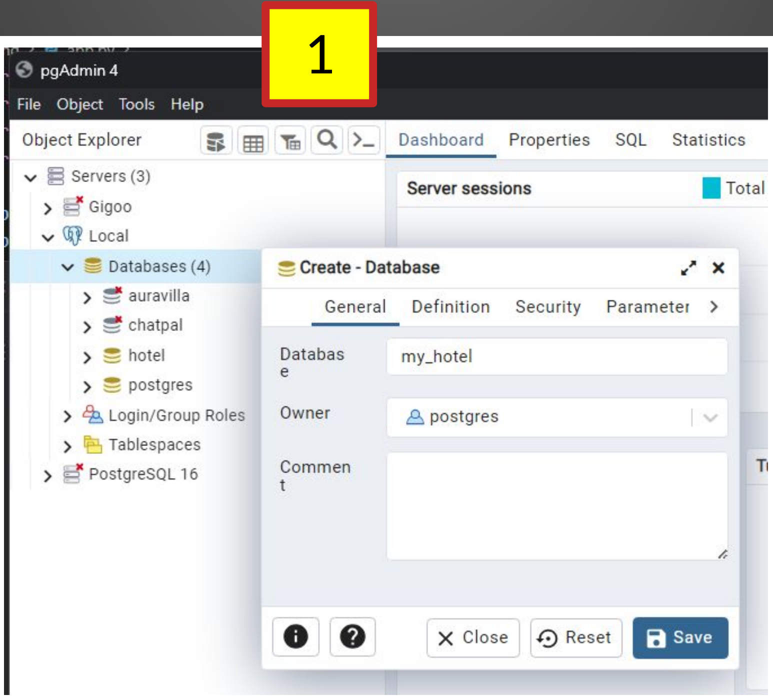
booking.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- [existing head content] -->
</head>
<body>
    <div class="container">
        <h2>Book a Room</h2>
        <form method="post">
            {{  form.hidden_tag() }}
            <div class="form-group">
                {{ form.guest_name.label }} {{
form.guest_name(class="form-control") }}
            </div>
            <div class="form-group">
                {{ form.room_number.label }} {{
form.room_number(class="form-control") }}
            </div>
            <div class="form-group">
                {{ form.check_in_date.label }} {{
form.check_in_date(class="form-control") }}
            </div>
            <div class="form-group">
                {{ form.check_out_date.label }} {{
form.check_out_date(class="form-control") }}
            </div>
            <div class="form-group">
                {{ form.contact_info.label }} {{
form.contact_info(class="form-control") }}
            </div>
            <div class="form-group">
                {{  form.submit(class="btn btn-primary") }}
            </div>
        </form>
    </div>
</body>
</html>
```

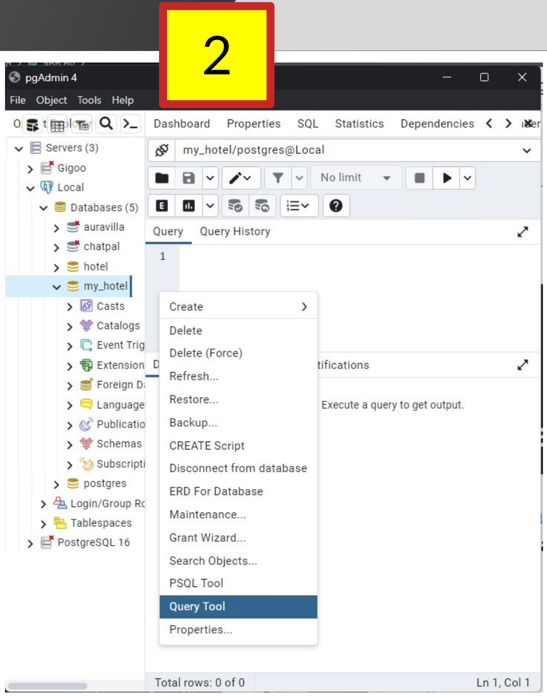
Setup Database

Create a Database

1. Create a database with a name
2. Open the pane of Query Tool.



Database: my_hotel
Owner: postgres
Comment:



Create tables

```
CREATE TABLE room (
    room_number VARCHAR(10) PRIMARY KEY,
    room_type VARCHAR(50),
    price_per_night DECIMAL(10, 2),
    max_guests INT
);
```

- **room_number**: The number or name of the room, unique for each room.
- **room_type**: Describes the type of room (e.g., Single, Double, Deluxe).
- **price_per_night**: The cost per night for the room.
- **max_guests**: The maximum number of guests allowed in the room.

```
CREATE TABLE guest (
    guest_id SERIAL PRIMARY KEY,
    guest_name VARCHAR(255) NOT
NULL,
    contact_info VARCHAR(255)
);
```

- **guest_id**: A unique identifier for each guest.
- **guest_name**: The name of the guest.
- **contact_info**: Contact information for the guest, such as a phone number or email address.

```
CREATE TABLE booking (
    booking_id SERIAL PRIMARY KEY,
    guest_id INT REFERENCES guest(guest_id),
    room_number VARCHAR(10) REFERENCES
room(room_number),
    check_in_date DATE,
    check_out_date DATE,
    total_price DECIMAL(10, 2),
    booking_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- **booking_id**: A unique identifier for each booking.
- **guest_id**: Foreign key linking to the guests table.
- **room_number**: Foreign key linking to the rooms table.
- **check_in_date**: The date when the guest will check in.
- **check_out_date**: The date when the guest will check out.
- **total_price**: The total price for the stay.
- **booking_date**: The timestamp when the booking was made (default to the current timestamp).

Add Rooms

```
insert into room (room_number, room_type, price_per_night, max_guests)
values ('101', 'Single Room', 1000, 2);

insert into room (room_number, room_type, price_per_night, max_guests)
values ('102', 'Double Room', 2000, 3);

insert into room (room_number, room_type, price_per_night, max_guests)
values ('103', 'Deluxe Room', 5000, 5)
```

Additional Considerations

- Indexes
  Depending on query patterns, you might want to add indexes on commonly queried columns like room_number, guest_name, or check_in_date and check_out_date in the bookings table.
- Data Validation
  Consider adding constraints and validations as needed. For instance, check-in and check-out dates should be logical (check-out should be after check-in).
- Normalization
  The schema is normalized to reduce redundancy. For example, guest information is stored separately from bookings to handle multiple bookings by the same guest efficiently.
- Security
  When implementing this schema in a real application, always be mindful of security practices, especially when handling personal data.

Integration

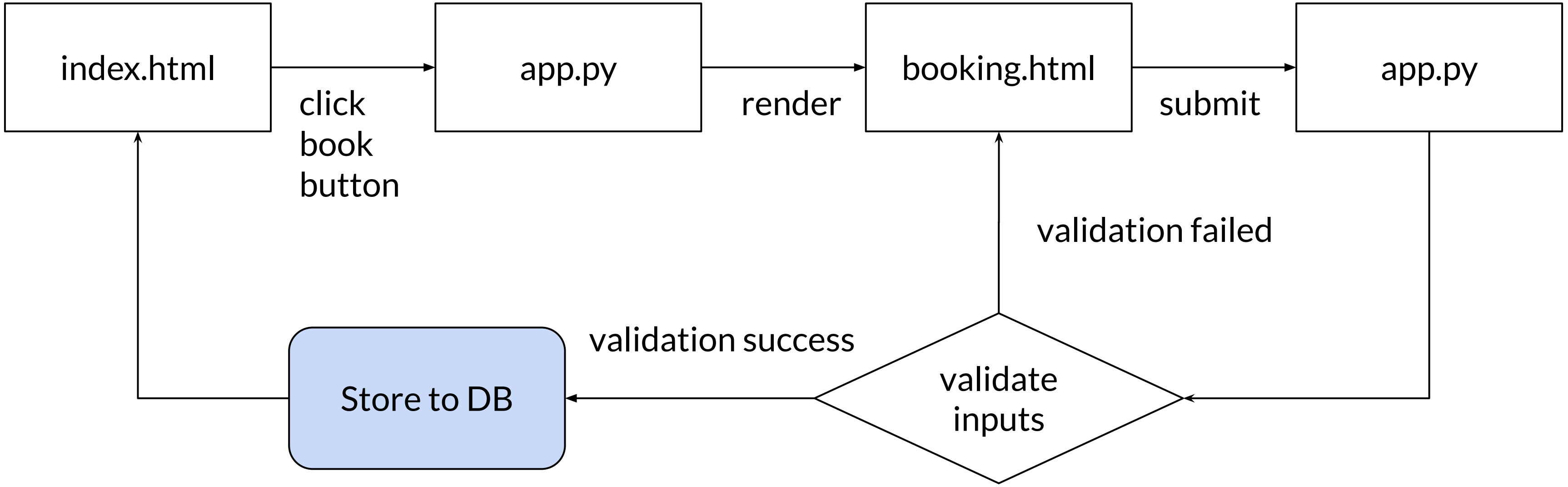
Update requirements.txt and reinstall

```
hotel_booking/
│
├── templates/
│   ├── index.html
│   └── booking.html
│
├── requirements.txt
└── app.py
```

```
Flask
Flask-WTF
WTForms
Flask-SQLAlchemy
psycopg2-binary
```

```
pip install -r requirements.txt
```

Processing Flow



app.py 1/2

```
from flask import Flask, render_template, request, redirect,
url_for
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField, DateField,
SelectField
from wtforms.validators import DataRequired

from flask import Flask, render_template, request, redirect,
url_for
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SECRET_KEY'] = 'YourSecretKey'
app.config['SQLALCHEMY_DATABASE_URI'] =
'postgresql://postgres:the_password@localhost/my_hotel'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

# Example room data (in a real application, this would come from a
database)
rooms = [
    ('101', 'Single Room'),
    ('102', 'Double Room'),
    ('103', 'Deluxe Room')
```

```
class Guest(db.Model):
    guest_id = db.Column(db.Integer, primary_key=True)
    guest_name = db.Column(db.String(255), nullable=False)
    contact_info = db.Column(db.String(255))

class Room(db.Model):
    room_number = db.Column(db.String(10),
primary_key=True)
    # other room fields...

class Booking(db.Model):
    booking_id = db.Column(db.Integer, primary_key=True)
    guest_id = db.Column(db.Integer,
db.ForeignKey('guest.guest_id'), nullable=False)
    room_number = db.Column(db.Integer,
db.ForeignKey('room.room_number'), nullable=False)
    check_in_date = db.Column(db.Date, nullable=False)
    check_out_date = db.Column(db.Date, nullable=False)
    # other booking fields...
```

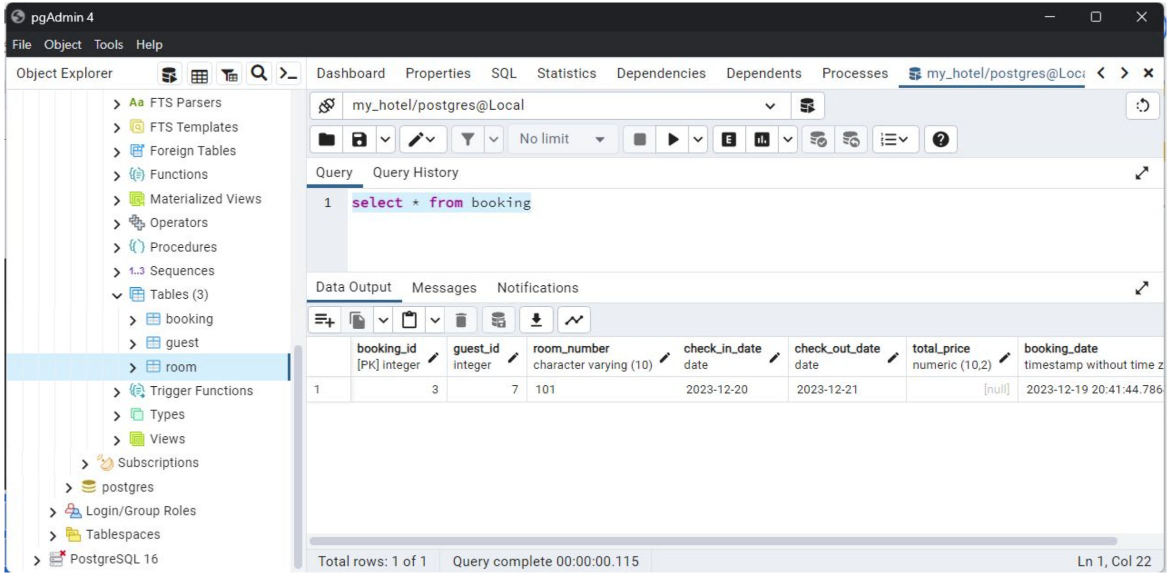
app.py 2/2

```
class BookingForm(FlaskForm):
    guest_name = StringField('Guest Name',
validators=[DataRequired()])
    room_number = SelectField('Room Number', choices=rooms,
validators=[DataRequired()])
    check_in_date = DateField('Check-In Date', format='%Y-%m-%d',
validators=[DataRequired()])
    check_out_date = DateField('Check-Out Date', format='%Y-%m-%d',
validators=[DataRequired()])
    contact_info = StringField('Contact Information',
validators=[DataRequired()])
    submit = SubmitField('Book Now')

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

```
@app.route('/booking', methods=['GET', 'POST'])
def booking():
    form = BookingForm()
    if form.validate_on_submit():
        new_guest = Guest(guest_name=form.guest_name.data,
contact_info=form.contact_info.data)
        db.session.add(new_guest)
        db.session.flush()  # Flush to get the ID of the
new guest

        new_booking = Booking(
            guest_id = new_guest.guest_id,
            room_number = form.room_number.data,
            check_in_date = form.check_in_date.data,
            check_out_date = form.check_out_date.data
        )
        db.session.add(new_booking)
        db.session.commit()

        return redirect(url_for('index'))

    return render_template('booking.html', form=form)

if __name__ == '__main__':
    app.run(debug=True)
```

Check the Result

```
select * from booking
```



```
select * from booking
```

	booking_id [PK] integer	guest_id integer	room_number character varying (10)	check_in_date date	check_out_date date	total_price numeric (10,2)	booking_date timestamp without time z
1	3	7	101	2023-12-20	2023-12-21	[null]	2023-12-19 20:41:44.786

Total rows: 1 of 1 Query complete 00:00:00.115 Ln 1, Col 22