Shiva Kasula - What I Built for This Project

My Role: Database & Logic Specialist
Project: School Activity Booking System

Quick Summary - What I Did

In Simple Words: I built the brain and memory of the system.
Think of it like this:
- I built the filing cabinet where everything is stored (Database)
- I built the rules that say "No, you can't book a full class" (Logic)
- I built the waiting line system (Waitlist)
My Main Jobs:
■ Designing the database (The Filing Cabinet)
■ Making sure bookings don't conflict (The Rules)
■ Handling payments (The Cash Register)
■ Managing the waitlist (The Queue)

Part 1: The Database (The Filing Cabinet)

What I Did (Simple Summary)

- Designed how we store all information
- Made sure everything is organized perfectly
- Connected things together (Parent ↔ Child ↔ Booking)

How Does It Work? (Easy Explanation)

Imagine a super organized office:
- Cabinet 1: **Parents** (Names, emails)
- Cabinet 2: **Children** (Names, ages)
- Cabinet 3: **Activities** (Swimming, Math)
- Cabinet 4: **Bookings** (Who booked what)
The Connections (Relationships):
- I tied strings between folders
- Parent folder has string to Child folder
- Booking folder has strings to Parent, Child, and Activity
Why?
- So we never lose track!
- If we look at a Booking, we can follow the string to find the Parent.

The Code (With Simple Explanation)

```
class Parent(db.Model):
id = Column(Integer)
name = Column(String)
# The string connecting to children
children = relationship('Child', backref='parent')
class Booking(db.Model):
id = Column(Integer)
# The strings connecting to everyone
parent_id = ForeignKey('parent.id')
child_id = ForeignKey('child.id')
```

activity_id = ForeignKey('activity.id')
Real Example:
We want to know: "Who booked Swimming?"
1. Go to Activity Cabinet -> Find "Swimming"
2. Follow string to Booking Cabinet -> Find 15 bookings
3. For each booking, follow string to Child Cabinet -> Read names
4. Result: "John, Emma, Mike..."

Part 2: The Booking Rules (The Bouncer)

What I Did (Simple Summary)

- Created rules to stop mistakes
- "Stop! This class is full!"
- "Stop! You already booked this!"
- "Stop! You can't be in two places at once!"

How Does It Work? (Easy Explanation)

Imagine a strict bouncer at the door of the class:
Rule 1: Capacity Check
- Class size = 20 students
- Bouncer counts people inside: 1, 2... 19, 20.
- Person 21 comes.
- Bouncer: "Sorry! Full! Go to waitlist."
Rule 2: Double Booking Check
- John tries to book Swimming (Monday 3pm)
- Bouncer checks list: "Wait, John is already on the list for Monday 3pm!"
- Bouncer: "You can't book twice!"
Rule 3: Time Travel Check
- Parent tries to book for Yesterday
- Bouncer: "We don't have a time machine! Future dates only!"

The Code (With Simple Explanation)

def book_activity():
# Rule 1: Is it full?
current_count = count_bookings(activity)
if current_count >= 20:
return "Sorry, Full!"
# Rule 2: Already booked?
if already_booked(child, activity, date):
return "You already booked this!"
# If passed all rules...
create_booking()
return "Success!"
Real Example:
Parent clicks "Book" for Swimming (Capacity 20)
Computer checks:
1. How many booked? -> 19
2. Is 19 < 20? -> Yes (Space available!)
3. Did this child book already? -> No
4. ■ Booking Approved!

Part 3: The Waitlist (The Queue)

What I Did (Simple Summary)

- Built a system for when classes are full
- If someone cancels, the next person gets in automatically
- First come, first served!

How Does It Work? (Easy Explanation)

Imagine a line outside a club:
- Club is full.
- 5 people waiting outside in a line.
- One person inside leaves.
- Bouncer waves to the *first* person in line: "You! Come in!"
My Automated System:
1. Class Full? -> Show "Join Waitlist" button
2. Parent clicks it -> Added to database with timestamp (Time they joined)
3. Someone cancels booking? -> System wakes up!
4. System checks waitlist -> Who joined first?
5. System promotes them -> Creates booking automatically!

The Code (With Simple Explanation)

def promote_from_waitlist(activity):
# Find the person waiting longest
next_person = Waitlist.query.filter_by(activity=activity)\
.order_by(time_joined)\
.first()
if next_person:
# Move them to booking
create_booking(next_person)
delete_from_waitlist(next_person)
send_email("You got a spot!")
Real Example:
1. Swimming is full.
2. Mom A joins waitlist at 10:00 AM.
3. Dad B joins waitlist at 10:05 AM.
4. Someone cancels at 2:00 PM.
5. System checks: Mom A was first.
6. System books Mom A automatically!
7. Dad B is now first in line.

Part 4: Payment Handling (The Cash Register)

What I Did (Simple Summary)

- Built the checkout process
- Calculate total cost
- Make sure payment happens *before* booking is confirmed

How Does It Work? (Easy Explanation)

Imagine a shop:
1. You pick item (Activity)
2. You go to counter
3. Clerk scans item -> "That's £25"
4. You pay
5. Clerk gives receipt (Booking Confirmation)
The "State Machine" (Steps):

- State 1: **Selection** (Picked activity)
- State 2: **Pending** (Entered card details)
- State 3: **Confirmed** (Payment success -> Booking created)
- State 4: **Failed** (Card declined -> No booking)

The Code (With Simple Explanation)

def process_payment():
# Step 1: Try to charge card
payment_success = charge_card(amount=25.00)
if payment_success:
# Step 2: Only create booking if paid!
create_booking()
return "Success"
else:
return "Payment Failed"
Real Example:
Parent clicks "Pay £25"
Computer:
1. Contacts Bank (Simulated)
2. Bank says "Approved"
3. Computer says "Great!"
4. Creates Booking in database
5. Shows "Success" page

My Contribution Summary

Files I Created/Modified:
1. `app.py` - Database models and booking logic (300+ lines)
2. Database Schema - The structure of all our tables
What Each Part Does (Simple):
Part	What It Does	Like...
Database	Stores all info	Filing Cabinet
Relationships	Connects data	Strings between folders
Validation	Checks rules	The Bouncer
Waitlist	Manages full classes	The Line Outside
Payment	Handles money	Cash Register

Why This Matters

Without my work:
- ■ Data would be messy and lost
- ■ 50 kids would book a class for 20 (Chaos!)
- ■ People could book without paying
- ■ Double bookings everywhere
With my work:
- ■ Organized information
- ■ No overbooking (Strict 20 limit)
- ■ Fair waitlist system
- ■ Secure payments

Shiva Kasula
Database & Logic Specialist

University of East London
December 2025