Sanchit Kaushal - What I Built for This Project

My Role: Team Leader & Security Expert
Project: School Activity Booking System

Quick Summary - What I Did

In Simple Words: I built the security system and admin control panel for our school booking website.
Think of it like this:
- I'm like the security guard who checks IDs at the door
- I built the manager's office where they control everything
- I made sure nobody can hack or break into the system
My Main Jobs:
■ Made sure passwords are safe
■ Built the login system
■ Created the admin control panel
■ Set up the website for internet deployment

Part 1: Making Passwords Safe

What I Did (Simple Summary)

- I made sure when you type your password, it gets scrambled up
- Nobody (not even us!) can see your real password
- Even if hackers steal our database, they can't read passwords

How Does It Work? (Easy Explanation)

Imagine this situation:
- You write "MyPassword123" on paper
- I put it in a super special blender
- It turns into "xY7#kL9@pQ2$" (scrambled mess)
- We store "xY7#kL9@pQ2$" in our safe
- **Nobody can unscramble it!**
When you login:
- You type "MyPassword123" again
- We scramble it the same way
- Does "xY7#kL9@pQ2$" match what's in the safe? ■ Login!
- Did you type wrong password? It scrambles different! ■ No entry!

The Code (With Simple Explanation)

def set_password(self, password):
# This is like putting password in the "super blender"
self.password = generate_password_hash(password)
What this means:
- `password` = What you typed (like "MyPassword123")
- `generate_password_hash` = The special blender
- `self.password` = The scrambled mess we save
Example:
You type: "ILoveCats2024"
Scrambled becomes: "scrypt:32768:8:1$aBc123xyz$9876..."
We save: The scrambled version (not the real one!)

Part 2: Login System

What I Did (Simple Summary)

- Built 3 different login doors (Parent, Admin, Tutor)
- Made sure only the right people can enter the right doors
- Like a nightclub with VIP sections!

How Does It Work? (Easy Explanation)

Think of our website like a building with 3 floors:
■ Building = Our Website
■■■ ■ Ground Floor Door = Parent Login
■ ■■■ Only parents with kids can enter
■■■ ■ 2nd Floor Door = Admin Login
■ ■■■ Only managers can enter
■■■ ■ 3rd Floor Door = Tutor Login
■■■ Only teachers can enter
What happens when you login:
Step 1: You arrive at the door
You: "Hi, I'm John and my password is ILoveCats2024"
Step 2: We check our list
Guard: "Let me check...
- Do we have 'John' in our book? ■ Yes!
- Does password match? ■ Yes!
- Here's your access card!"
Step 3: We give you a magic bracelet
- We put invisible bracelet on you
- Every time you click something, we check: "Does this person have bracelet?"
- If yes = You can see the page!
- If no = "Sorry, please login first!"

The Code (With Simple Explanation)

@app.route('/login', methods=['POST'])
def login():
# Step 1: Get what user typed
email = request.form.get('email')
password = request.form.get('password')
# Step 2: Check our database (like looking in a phonebook)
parent = Parent.query.filter_by(email=email).first()
# Step 3: Does password match?
if parent and parent.check_password(password):
# Step 4: Give them the magic bracelet!
session['parent_id'] = parent.id
return redirect('/dashboard') # Let them in!
else:
flash('Wrong email or password!') # Show error message
Real Example:
User types:
Email: john@example.com
Password: ILoveCats2024
Computer checks:
1. Is john@example.com in database? ■
2. Does "ILoveCats2024" scrambled match saved scrambled? ■
3. Give session bracelet: session['parent_id'] = 42
4. Send to dashboard page!

Part 3: The Admin Control Panel

What I Did (Simple Summary)

- Built a special control room for managers
- They can create/edit/delete activities
- Like a TV remote control - but for the whole website!

How Does It Work? (Easy Explanation)

Imagine the admin panel like a car dashboard:
■ Admin Dashboard = Control Panel
■■■ ■ Speedometer = See how many bookings today
■■■ ■ Fuel Gauge = See how much money earned
■■■ ■■ Buttons = Create new activities
■■■ ■■ Trash = Delete old activities
What admins can do:
1. See Statistics (Like checking your phone battery)
Total Bookings Today: 47
Total Money Earned: £1,250
Active Activities: 8
2. Create New Activity (Like adding new contact in phone)
Admin clicks "New Activity"
Fills form:
- Name: "Swimming Lessons"
- Price: £25
- Max Students: 15
- Day: Monday
- Time: 3:00 PM - 4:00 PM
Clicks "Save"
■ New activity appears on website!
3. Edit Activity (Like editing a contact)
Admin sees: "Swimming - £25"
Clicks "Edit"
Changes price to: £30
Clicks "Save"
■ Updated!
4. Delete Activity (Like deleting old photo)
Admin clicks "Delete" on old activity
Computer asks: "Are you sure?"
Admin clicks: "Yes, delete it"
■■ Gone forever! (and all its bookings)

The Code (With Simple Explanation)

Creating New Activity:
@app.route('/admin/create_activity', methods=['POST'])
def create_activity():
# Get what admin typed in the form
name = request.form.get('name') # "Swimming"
price = request.form.get('price') # "25.00"
max_capacity = request.form.get('max_capacity') # "15"
# Create new activity (like creating new contact)
new_activity = Activity(
name=name,
price=price,
max_capacity=max_capacity

)
# Save to database (like saving contact to phone)
db.session.add(new_activity)
db.session.commit() # ■ Saved!
return "Activity created!"
Real Example:
Admin fills form:
Name: "Art Class"
Price: £20
Capacity: 10
Day: Friday
Time: 2:00 PM
Computer does:
1. Create new_activity with all these details
2. Save to database
3. Show success message: "Art Class created! ■"
4. Now parents can see "Art Class" and book it!

Part 4: Website Security Guards

What I Did (Simple Summary)

- Built security guards that check every page
- Made sure parents can't access admin pages
- Like having bouncers at a VIP party!

How Does It Work? (Easy Explanation)

Imagine each webpage has a security guard:
Regular Page (Anyone can visit):
■■■■■■■■■■■■■■■■■
■ Home ■ No guard needed
■ Page ■ Everyone welcome!
■■■■■■■■■■■■■■■■■
Protected Page (Need login):
■■■■■■■■■■■■■■■■■
■ Dashboard ■
■ Page ■ ■ Guard checks: "Show me your bracelet!"
■■■■■■■■■■■■■■■■■
Super Protected (Only admins):
■■■■■■■■■■■■■■■■■
■ Admin ■
■ Panel ■ ■■■■■■■■ TWO guards: "Are you the boss?"
■■■■■■■■■■■■■■■■■
What happens when you try to visit protected page:
Scenario 1 - You're logged in:
You: Click "My Dashboard"
Guard: "Show me your bracelet"
You: *shows session bracelet*
Guard: "Ok, you're logged in! Enter! ■"
Scenario 2 - You're NOT logged in:
You: Click "My Dashboard"
Guard: "Show me your bracelet"
You: *no bracelet*
Guard: "Sorry! Go to login page first! ■"
Computer sends you to login page
Scenario 3 - Wrong type of user:

Parent: Tries to access Admin Panel
Guard: "Your bracelet says 'Parent', this is 'Admin Only'!"
Parent: ■ Access Denied!

The Code (With Simple Explanation)

The Security Guard (Decorator):
def login_required(f):
# This is like hiring a security guard
def check_before_entering(*args, **kwargs):
# Guard checks: Do you have login bracelet?
if 'parent_id' not in session:
# No bracelet = send to login page
return redirect('/login')
Has bracelet = let them in!
return f(*args, **kwargs)
return check_before_entering
Using the Guard:
@app.route('/dashboard')
@login_required # ← Security guard stands here!
def dashboard():
# Only people who pass the guard can reach this code
return "Welcome to your dashboard!"
Real Example:
What happens step-by-step:
1. User clicks "Dashboard" link
2. @login_required guard activates
3. Guard checks: session['parent_id'] exists?
If YES (logged in):
- Guard: ■ "Go ahead!"
- User sees dashboard
If NO (not logged in):
- Guard: ■ "Stop! Login first!"
- redirect('/login')
- User sent to login page

Part 5: CSRF Protection (Super Important Security!)

What I Did (Simple Summary)

- Protected against sneaky hackers
- Made sure forms can't be faked
- Like putting a special seal on official documents

How Does It Work? (Easy Explanation)

Imagine this bad situation WITHOUT protection:
You login to our website ■
You visit evil website (still logged in) ■
Evil website has hidden form that says:
"Delete John's account on School Website"
Your browser automatically sends it (because you're logged in!)
■ Your account deleted!
How we prevent this:
Step 1: We give you a secret code
When you visit our website:
- We create random code: "abc123xyz"
- We remember this code

- We put it in all our forms (hidden)
Step 2: When you submit form
You click "Book Activity"
Form includes:
- Activity: Swimming
- Child: Emma
- CSRF Code: "abc123xyz" ← Secret code!
Step 3: We check the code
Form arrives at server
Guard checks: "Does code 'abc123xyz' match what we gave earlier?"
YES = ■ This is real form from our website!
NO = ■ Fake form from hacker! REJECT!
Why evil website can't fake it:
Evil website tries to submit form
But they don't know our secret code "abc123xyz"
Their form has wrong code or no code
■ We reject it! Attack blocked!

The Code (With Simple Explanation)

In HTML form:
{{ csrf_token() }}
Book Activity
What user sees:
Book Activity
When form is submitted:

Automatic check happens before your code runs

If CSRF token wrong = Error before reaching here

@app.route('/book', methods=['POST'])
def book_activity():
# If code reaches here, CSRF check already passed! ■
activity = request.form.get('activity')
# Process booking...

Part 6: Deployment Setup

What I Did (Simple Summary)

- Prepared website to go on the internet
- Like packing a suitcase for a trip
- Made sure it works on real servers (not just my computer)

How Does It Work? (Easy Explanation)

Running on my computer vs Real internet:
My Computer (Development):
■ Like testing recipe in home kitchen
- I can see all the ingredients
- I can change recipe anytime
- Only I can taste it

Real Internet (Production):
■ Like restaurant serving customers
- Professional kitchen
- Many people eating at once
- Need proper equipment
- Can't just stop and change recipe!
What I prepared:
1. Requirements List (Like shopping list)
File: requirements.txt
Flask==2.3.0 (Main framework - like oven)
Flask-Mail==0.9.1 (Email system - like delivery service)
ReportLab==4.0.4 (PDF maker - like printer)
2. Instructions for Server (Like cooking instructions)
File: Procfile
web: gunicorn app:app
Translation: "Run the website using professional server software"
3. Secret Settings (Like safe combination)
File: .env
SECRET_KEY=super-secret-only-i-know
MAIL_PASSWORD=email-app-password
DATABASE_URL=where-to-save-data
Think of it like moving:
My Computer:
■ Everything in one box
■ Small apartment (SQLite database)
■ Just me using it
Real Server (Render/Heroku):
■ Organized in proper boxes
■ Big warehouse (PostgreSQL database)
■ Hundreds of people using it at once

My Contribution Summary

Files I Created/Modified:
1. `app.py` - Added security, login, admin features (300+ lines)
2. `config.py` - Settings file (complete)
3. `Procfile` - Deployment instructions
4. `.env.example` - Secret settings template
5. Admin templates - Control panel pages
What Each Part Does (Simple):
Part	What It Does	Like...
Password Hashing	Scrambles passwords	Putting paper in blender
Login System	Checks who you are	Security guard checking ID
Sessions	Remembers you're logged in	Invisible bracelet
CSRF Protection	Stops fake forms	Checking document seal
Admin Panel	Control everything	TV remote for website
RBAC Decorators	Allow/Block pages	Bouncers at VIP club
Deployment Config	Run on internet	Packing for trip

Why This Matters

Without my security work:
- ■ Hackers could steal passwords
- ■ Anyone could access admin panel

- ■ Evil websites could do fake actions
- ■ Website wouldn't work on internet
With my security work:
- ■ Passwords super safe (even we can't see them)
- ■ Only right people access right pages
- ■ Protected against hacker attacks
- ■ Ready for real internet use
- ■ Thousands can use it safely

Real-World Impact

Example Day in Life of My Code:
9:00 AM - Parent tries to login
- My password check: ■ Correct! Let them in!
10:30 AM - Hacker tries to fake a form
- My CSRF protection: ■ Blocked! Nice try!
2:00 PM - Parent tries to access admin panel
- My security guard: ■ "Sorry, admins only!"
3:45 PM - Admin creates new activity
- My admin panel: ■ Created successfully!
All Day - Website running on internet
- My deployment config: ■ Serving 100+ users smoothly!

Sanchit Kaushal
Team Leader
University of East London
December 2025