# Shift Scheduling and Automation System – Technical Guide

**Shift Scheduling and Automation System**
**Technical Guide**

**Student Name: Lucas Harper**
**Student ID: 21331096**
**Supervisor: Prof. Boualem Benatallah**
**Date: 01/05/2025**

## Abstract:

**This project presents a real-time shift scheduling and automation system designed for high-pressure environments like airport security. Built using React, Django, and GPT-4, the system manages break assignments, monitors operational coverage, and simulates real-time shift progression through a custom test clock. By integrating AI and domain-specific rules, it reduces scheduling errors, ensures legal compliance, and improves overall efficiency for coordinators managing large staff rosters.**

## Contents

## 1. Motivation

**I've been working as an Airport Search Unit (ASU) Officer at Dublin Airport for the last two years. It's a busy, hands-on job where you're dealing with passengers, queues, and tight security checks every day. What I noticed early on, though, was how much pressure falls on the coordinators — the people in charge of figuring out who goes on break, what machine is covered, and how to make sure everything keeps moving with hundreds of officers on shift.**

**What surprised me was how manual the whole process still was. Coordinators were using Excel sheets or even jotting things down on paper to manage the day — and when you're working in an environment that intense, that kind of system feels like it's just asking for something to go wrong.**

**As I got to know the job better, I started talking more with supervisors and duty managers. The same problems kept coming up: breaks being late, staff going over their allowed hours, machines uncovered — mostly because the tools they had weren't built for the complexity of the job.**

**That's when I realised there was something I could do. I'm in my final year of Computer Science, and this seemed like the perfect project to apply what I've learned — building a smarter, real-time system that takes the stress off coordinators and avoids the usual errors. My goal wasn't just to make a digital version of what already existed, but to build something better: a tool that uses AI to suggest break times, monitor shift progress, and help staff make quicker, smarter decisions on the ground.**

**That idea turned into this project: a full-stack shift scheduling and automation platform using React, Django, and OpenAI's GPT — designed from the ground up for the kind of real-world challenges I saw every day at the airport.**

---

## 2. Research

**I wanted this project to bridge two things I know well: how airport shifts work in real life, and how to build software that actually helps. From the start, I wasn't just trying to build something technical — I wanted to create a tool that would feel familiar and useful to the people doing the job every day, especially the coordinators and managers I worked with.**

### 2.1 Tools I used — React, Django, OpenAI GPT

**I went with React for the frontend because I've used it throughout my degree and in side projects. Its component system made it easy to manage the live updates I needed — like shift countdowns, break timers, and the AI assistant alerts that react to time changes. Hooks and state management helped keep everything smooth as people moved between On Duty, On Break, and Finished.**

**The backend runs on Django, which I first learned in college but became more confident with while working on personal and group projects. It gave me a solid base for building out the API, connecting the frontend to the backend logic, and handling requests to OpenAI's GPT engine.**

**For the AI part, I used GPT-4 to help plan breaks. Getting that to work wasn't just about calling the API — I had to dig into how prompts are structured, figure out token limits, and make sure the output format (like {ID: BreakTime} or {ID: [First, Second]}) could be plugged right into my system. The goal was to have the AI suggest smart, rule-following break times that synced perfectly with the test clock.**

## 2.2 Understanding the Operational Rules

**A lot of what makes this system work came from simply paying attention on the job. Talking to coordinators and duty managers gave me a clearer picture of the rules they deal with — and the ones they sometimes have to bend in emergencies. Things like:**

- Staff are entitled to one or two breaks depending on shift length.
- No one should go over 4.5 hours without a break.
- At least 93 people must be on duty at all times, except when someone is due a break, when it can drop temporarily to 74.
- Staff are tagged by gender and role (e.g. X-Ray, WTMD), and specific coverage requirements must always be maintained (e.g. a male required for certain positions).
- Machines cannot be left unattended — if someone goes on break, they must be replaced, or another machine must be closed safely.

**I built my data model around these ideas. For example, each person has a shift start and end time, a count of how many breaks they've taken, and a gender tag. The AI uses that info — plus the current time — to decide who should go next. The co-pilot even gives warnings if someone is coming close to the 4.5-hour limit.**

## 2.3 Data Handling & Break Scheduling Logic

**My internship at Davy taught me a lot about dealing with messy data and scaling things up properly. I brought that mindset here.**

**Since large rosters can have hundreds of people, I split GPT queries into chunks — first 100, next 200, and so on — to stay under token limits. I also made sure once someone gets a break time, it's locked in. The AI won't assign a new one unless it has to. That way, the schedule stays stable.**

**I also added validation to block third breaks or overlaps. Breaks come in a format like:**
100023 06:10 09:30
**and the system checks each one to make sure it's allowed.**

**In full automation mode, the system can:**

1. Automatically move people from Rollcall to On Duty when their shift starts
2. Send their data (plus passenger traffic) to GPT every two hours
3. Trigger breaks when the test clock hits the assigned time
4. Move people to Finished when their break time is over and their shift is done

**A lot of trial and error went into getting this right — but it helped that I had real rules from the job to guide me, plus the tech skills from college and previous work to build the system in a way that holds up.**

---

## 3. Design

**When I started designing this system, I wanted it to reflect the real pressure and decision-making I'd seen during my time at the airport — not just look good on paper. The idea was to build something that mimics how coordinators actually think about shifts, breaks, and coverage, while also giving them tools they don't have right now — like AI support and live updates.**

**The project is built around five main parts:**

1. A live React frontend to manage shifts
2. A Django backend that runs the API and business logic
3. A break scheduler powered by GPT-4
4. A test clock that simulates time passing
5. Rule enforcement to make sure all decisions follow operational policies

---

### 3.1 Frontend Design — React

**The frontend is built with React and split into three core areas that match how shifts are tracked in real life:**

- On Duty: Shows staff currently working, with a progress bar that tracks how far into their shift they are.
- On Break: Lists who's on break, with a live countdown until they return.
- Finished: For people who've either finished their shift or taken their full set of breaks.

**Each group is powered by a state array (onDutyProducts, onBreak, finished) and updates automatically when someone changes state — like when the AI assigns a break, the test clock hits a key time, or the user manually adjusts something.**

**A few other features help make the interface easy to use and powerful:**

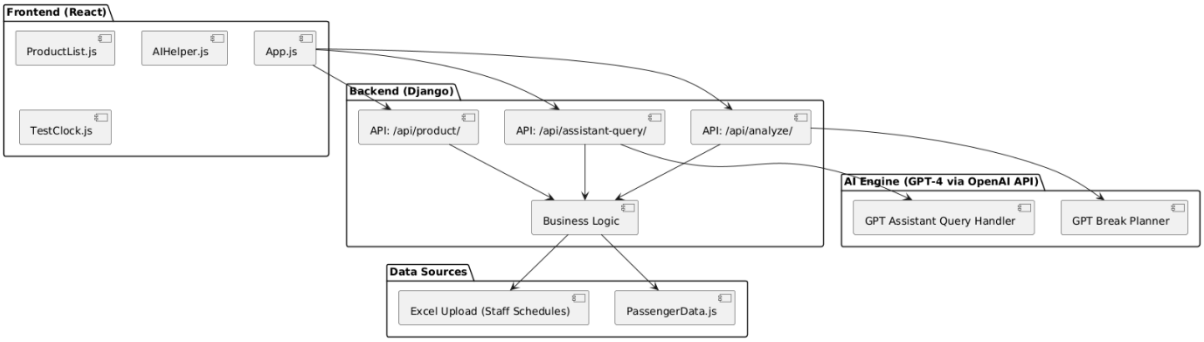
- **Test Clock**: This replaces the real system time so I can simulate different parts of the day instantly. Really useful for testing shift progress and break logic.
- **AI Co-Pilot**: A smart assistant powered by GPT-4 that pops up alerts like "100002 is due a break soon" or "coverage might drop below 93".

- **Chat Assistant**: You can type in natural language questions like “Who can I finish early?” and it’ll respond using real data from the shift.
- **Sorting Tools**: Coordinators can sort On Duty staff by name, shift start, or end — all in a collapsible panel that keeps the layout clean.
- **Person Cards**: Each staff member is shown in a styled card with name, shift info, a visual progress bar, and break history (based on how many breaks they’ve taken).

---

## 3.2 Backend Design — Django

**The backend is built in Django and handles all the behind-the-scenes logic. It connects the frontend to the GPT scheduling engine and makes sure everything follows the rules.**




**I set up REST API endpoints for tasks like:**

- Sending the current shift data to GPT for analysis
- Receiving break schedules and passing them back to the frontend
- Letting users type natural language queries for the assistant

**It also keeps track of:**

- Each person’s assigned break times ({ID: breakTime} or {ID: [First, Second]})
- Making sure people don’t get new break times if they already have them
- Automatically moving someone to "Finished" once their breaks are done and they’re eligible to go home

**I added validation throughout to prevent weird edge cases — like a third break being assigned or someone getting a break too close to the end of their shift.**

---

## 3.3 Break Scheduling Engine — OpenAI GPT-4 Integration

**One of the biggest challenges — and probably the most interesting part — was getting GPT-4 to handle break scheduling.**

**I had to figure out how to get the AI to understand airport rules the way a coordinator does. I built a detailed prompt that includes:**

- The current On Duty staff (with their role, gender, break history, etc.)
- Who's already on break
- A snapshot of passenger traffic (green = low, red = peak)
- The operational rules (like minimum staff on duty, role coverage, max two breaks, etc.)

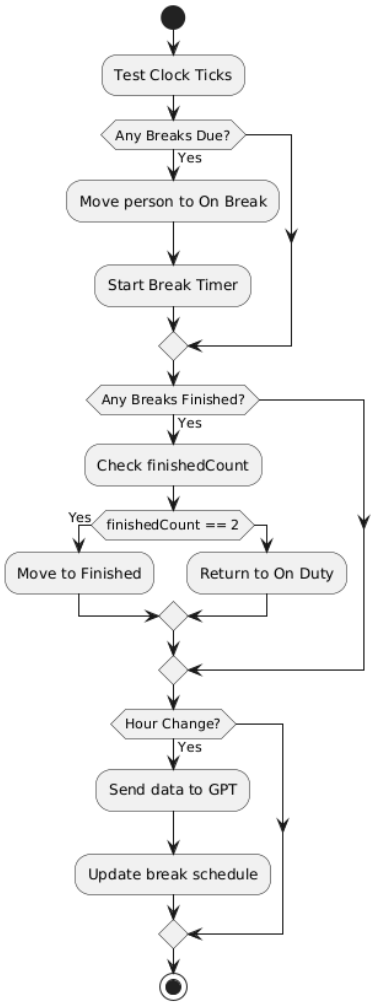
**I designed the output format to be simple and usable right away — something like:**

```
{
  "100004": "06:10",
  "100021": "07:00"
}

```

**Or, if two breaks:**

```
{
  "100418": ["05:20", "09:40"]
}

```

**That structure allowed me to sync AI decisions directly with the test clock. When the test time matched a scheduled break time, the person would automatically move to "On Break".**




---

## 3.4 Automation Engine — Test Clock Driven

**Instead of relying on real time (which would take forever to test), I built a custom test clock that simulates time passing minute by minute.**

**Everything in the system runs off this clock:**

- Shift progress bars update based on it
- AI Co-Pilot alerts are triggered based on how long someone's been on duty
- Breaks are triggered exactly when their scheduled time matches the clock
- People are moved to "Finished" once their second break is over (or if their shift ends)

**Every hour (test time), the system sends a fresh batch of shift data to GPT for analysis. That way, it always has the latest info when making decisions.**

---

### 3.5 Operational Rule Handling

**Most of the logic behind this system came straight from conversations I had on the job — with supervisors, coordinators, and the IT team. I didn't want to just hard-code generic rules; I wanted to reflect what actually happens on the ground.**

**Here are some examples of how real policies are built into the app:**

| Rule/Constraint | Implementation |
|---|---|
| **Must be 93 on duty at all times** | Checked before any break/finish is triggered |
| **Can drop to 74 only if break due** | AI engine allows it only for urgent breaks |
| **Must replace X-Ray / WTMD roles** | AI is prompted with role requirements |
| **Males needed for certain positions** | GPT considers gender in replacement decisions |
| **One or two breaks only** | finishedCount and break history enforced |
| **No break within first hour of shift** | Prevented by test clock filter logic |
| **Second break must be at least 2h after first** | Validated by comparison in mapping store |
| **Rule/Constraint** | Implementation |

**The system doesn't just "follow the rules" — it reflects the kind of thinking that goes into real shift planning at the airport.**

---

## 4. Implementation

**I built this project as a full-stack web app using React for the frontend and Django for the backend. The two parts talk to each other through clean REST API endpoints, and everything is wired up to support real-time shift tracking, AI break scheduling, and automated updates as time moves forward.**

**The whole thing is built to be modular and flexible — so different parts like the AI co-pilot, test clock, or break logic can be changed without having to rebuild everything else.**

### 4.1 Frontend (React.js)

**The frontend is implemented in JavaScript using the React framework. The key features include:**

**Component Architecture**

- App.js: The root component, which manages the state for onDuty, onBreak, and finished staff groups and the current test clock.

- Product.js: Displays individual staff cards in each shift section (e.g., On Duty, On Break), and manages local logic such as break timers and shift progress.
- ProductList.js: Handles sorting and rendering the different staff groups using dropdown menus and icons.
- AIHelper.js: Manages proactive AI co-pilot logic, including alerts such as “break due soon”, “finish early”, and “insufficient coverage”.
- AIAssistant.js: Implements a text-based assistant that lets users ask questions like "Who is next due a break?" or "Who can I finish early today?" and receive natural language responses using GPT.

**Styling & Interactivity**

- All CSS files are scoped per component (App.css, AIHelper.css, etc.) to ensure modular styling.
- PopoutMenu.js and PopoutMenu.css create the AI co-pilot UI, including a circular toggle button that expands into notifications or assistant panels.

**Clock System**

- A test clock is implemented which acts as the system time, allowing realistic testing of shift activity over time.
- All break countdowns and shift progress bars are synchronized to this the test clock.

**Data Handling**

- analyzeShifts.js handles POST requests to /analyze/, sending shift data and receiving GPT-generated break assignments.
- passengerData.js provides a local simulation of hourly passenger traffic used to inform break planning.

---

## 4.2 Backend (Django)

**The Django backend is where most of the logic lives — handling break scheduling, GPT prompts, validation, and API communication.**

**I created a few key endpoints:**

- GET /product/: Returns a filtered and deduplicated list of active staff scheduled in Terminal 1, excluding certain early shifts.
- POST /assistant-query/: Accepts a natural language query and current staff state, sends a formatted prompt to GPT, and returns the AI’s advice.

- POST /analyze/: Sends on-duty and break data along with passenger load to GPT, and receives structured break assignments in the format {ID: {first, second}}.

**The backend uses the OpenAI SDK to craft prompts that reflect actual airport scheduling rules — including timing gaps, role coverage, break caps, and more.**

- Shift start/end
- Minutes worked
- Breaks taken
- Ideal break windows
- Passenger load traffic data

**Two different GPT prompts are used:**

- One for user queries (assistant_query)
- One for full shift analysis (analyze_shifts)

**I baked in a bunch of rules that mirror what real coordinators deal with:**

- First breaks have to happen between 2 and 4.5 hours into a shift
- Second breaks must be at least 2 hours after the first and end at least 40 minutes before the shift ends
- No more than 20 people are allowed per break slot (to prevent overload)
- There must always be at least 93 staff on duty — unless someone urgently needs a break, in which case it can drop to 74 briefly

**GPT is prompted with all of these limits so that it can act like a smart, rule-following coordinator — rather than just picking random break times.**

---

## 4.3 Technologies Used

| Layer | Technology |
|---|---|
| **Frontend** | React, JavaScript, CSS |
| **Backend** | Django, Django REST |
| **AI Engine** | OpenAI GPT (o4-mini) |
| **API** | REST over HTTP |
| **Environment** | Python, Node.js, .env |

## 4.4 Continuous Deployment

**To ensure that the system remains up-to-date and testable throughout development, a Continuous Deployment pipeline was implemented using Render for the backend and Vercel for the frontend.**

**The Django backend is deployed on Render (Link to Render), which monitors the GitHub repository for changes to the main branch. When changes are detected, Render automatically pulls the latest code, builds the environment, and restarts the application using environment variables for sensitive configuration such as API keys and allowed hosts.**

**The React frontend is deployed on Vercel(Link to Vercel), with the live application accessible at https://2025-csc1097-lharper2.vercel.app/. Vercel also listens to commits on the main branch and performs automatic builds and deployments when updates are pushed. This enables real-time access to the latest UI features for demonstration and testing purposes.**

**Together, these deployment platforms ensure the system is always running the latest tested version, with minimal manual intervention. This CD setup also makes it easier to share and validate the application with Supervisors during the development cycles.**

---

# 5. Sample Code

**Here are a few examples from the project that show how everything fits together — from sending data to the AI, to building GPT prompts, to triggering real-time break alerts in the UI.**

---

## 5.1 Frontend – Sending Data to Analyze Shifts

**This function (analyzeShifts.js) sends the current On Duty and On Break staff, plus passenger traffic and the current hour, to the backend. That backend then builds a GPT prompt and gets back recommended break times.**

**Here's the code that handles that call:**

```
1. // analyzeShifts.js
 2. export const analyzeShifts = async (onDuty, onBreak, passengerData, currentHour) => {
 3.   try {
 4.     const response = await fetch('/api/analyze/', {
 5.       method: 'POST',
 6.       headers: {
 7.         'Content-Type': 'application/json',
 8.       },
 9.       body: JSON.stringify({ onDuty, onBreak, passengerData, currentHour }),
10.     });
11.
12.     return await response.json();
```

```
  } catch (error) {
    console.error("Error analyzing shifts:", error);
    return {};
  }
};

```

**This made it easy to trigger break suggestions from anywhere in the app without repeating the same fetch logic.**

---

## 5.2 Backend – Constructing GPT Prompt for Break Assignment

**On the backend, I take the shift and passenger data and build a big text prompt for GPT to read — like giving it a live status report.**

**That looks something like this:**

```
# views.py
prompt = (
    f"You are an AI shift scheduling assistant for an airport.\n\n"
    f"The current hour is {current_hour}.\n\n"
    f"On Duty Staff:\n{format_shift_data(on_duty)}\n\n"
    f"On Break:\n{format_shift_data(on_break)}\n\n"
    f"Passenger traffic status:\n{format_traffic(passenger_data)}\n\n"
    "Your job is to assign both a **first** and **second** break time to every staff member..."
)

```

**GPT then replies with break times in a format like this:**

100203 06:10 09:30

100218 07:00

**I parse that back into usable data using this code:**

```
lines = output.splitlines()
schedule = {}

for line in lines:
    parts = line.strip().split()
    if len(parts) == 3:
        staff_id, first_break, second_break = parts
        schedule[staff_id] = { "first": first_break, "second": second_break }
    elif len(parts) == 2:
        staff_id, first_break = parts
        schedule[staff_id] = { "first": first_break }

```

**Once it's parsed, I send it back to the frontend where it gets matched to the correct person and scheduled live.**

---

## 5.3 Frontend – Co-Pilot Break Warning Alert

**This hook inside AIHelper.js constantly checks if any On Duty staff are getting close to the 4.5-hour limit without having taken a break. If someone hits that window, it pops up a warning to remind the coordinator.**

```
// AIHelper.js
useEffect(() => {
  onDuty.forEach(person => {
    const shiftStart = parseTime(person.Shift_Start_Time);
    const minutesWorked = (testClockHour * 60 + testClockMinute) - (shiftStart.hours * 60 +
shiftStart.minutes);

    if (minutesWorked >= 270 && person.finishedCount === 0) {
      addAlert(`${person.name} is due a break soon!`);
    }
  });
}, [testClockHour, testClockMinute, onDuty]);

```

**This was one of the simplest features to implement — but it's also one of the most useful, since it stops people from accidentally going over their legal working time without a break.**

---

## 6. Problems Solved

**While building the system, I ran into a bunch of real-world issues — both technical and operational — that I needed to figure out along the way. Most of these weren't things you'd spot in a spec document; they came from being on the ground and seeing how chaotic shift management can get. Here's a breakdown of some major problems I tackled and how I solved them.**

---

### 6.1 Manual Scheduling Inefficiency

**Problem:**
Coordinators were using Excel sheets or handwritten notes to manage breaks, coverage, and shift flow. It was time-consuming, error-prone, and hard to update if something changed mid-shift. Important rules like machine coverage or gender-specific roles were often just managed by memory.

**Solution:**
Since I'd spent time actually working in the terminal, I saw how often things went wrong due to outdated info or simple oversight. I designed a live system where all shifts are visual, updated in real time, and backed by AI suggestions. Now the platform handles logic like "who needs to cover X-Ray" or "is a male officer needed here?" automatically — instead of leaving that pressure on the coordinator's head.

---

### 6.2 Coordinating Complex Break Rules

**Problem:**
It was tough for coordinators to stay on top of all the overlapping rules — like giving breaks before the 4.5-hour mark, keeping 93+ staff on the floor, and making sure machines weren't left uncovered.

**Solution:**
I embedded all of those rules directly into the AI prompt and backend validation logic. GPT gets a list of who's working, what roles they're in, and the passenger traffic load — and returns a schedule that respects the rules. Each break is checked to make sure it fits within time limits and doesn't cause any coverage issues.

---

## 6.3 Real-Time Break Reminders and Staff Monitoring

**Problem:**
In real operations, it's easy to lose track of time. People were sometimes working past the 4.5-hour limit without a break — not out of neglect, but because no one was watching the clock that closely during busy periods.

**Solution:**
I built the AI Co-Pilot system to watch for that. It checks how long each person has been on duty (using the test clock), and if someone's getting close to the limit without a break, it sends a pop-up alert. That kind of real-time nudge helps keep the shift compliant without adding extra stress for supervisors.

---

## 6.4 Ensuring Shift Coverage With Variable Demand

**Problem:**
During high-traffic times, giving breaks or finishing staff early was risky — it could lead to understaffing, especially if multiple roles were left uncovered by accident.

**Solution:**
I used a traffic schedule (passengerData.js) that flags low-load ("green") and high-load ("red") hours. GPT was prompted to prefer break assignments in green zones and avoid sending people on break when traffic was peaking. That helped spread breaks more evenly and kept the floor properly staffed during crunch time.

---

## 6.5 Automating Finishes and Shift Lifecycle Transitions

**Problem:**
Even after people had taken all their breaks or finished their shifts, they were sometimes still listed as "on duty" because no one manually moved them. This made the interface cluttered and caused mistakes during handovers.

**Solution:**
I added logic that automatically moves people to the "Finished" section once they've taken two breaks and met all coverage rules. That happens automatically in full automation mode — no button presses needed. It keeps the dashboard clean and clearly shows who's still active.

### 6.6 Handling Overnight Shifts and Invalid Time Formats

**Problem:**
Some shifts spanned midnight (e.g., 21:00 to 05:00), which broke the time logic by creating negative durations. I also hit crashes when time strings were missing or formatted wrong.

**Solution:**
I updated the backend to handle rollover shifts by checking if the end time is before the start time and adjusting the math accordingly. I also added error handling that catches invalid times and skips over broken entries, logging a note instead of crashing the whole system.

---

## 7. Results

**Once the system was fully built, I tested it using real-world scenarios based on my experience at the airport. The goal was simple: can it keep up with a live shift, follow all the break rules, and help reduce the mental load on coordinators? The results were really promising — it handled large rosters, followed the rules, and gave accurate, staggered break times even under heavy load.**

---

### 7.1 Operational Accuracy

**The system successfully handled all primary requirements related to break scheduling and shift flow management. Key outcomes include:**

- **Compliance with Break Timing Rules**
  Every person got their first break before hitting the 4.5-hour limit. Second breaks were spaced properly and finished before the end of the shift — all automatically calculated and enforced.
- **Minimum Coverage Constraints Maintained**
  The AI never dropped staff below 93 on duty unless a break was urgently required in those cases, it allowed a temporary dip to 74, just like actual ops do.
- **Staggered Break Distribution**
  Breaks were spaced in clean 10-minute intervals (e.g., 06:00, 06:10, 06:20…), and no more than 20 people were ever assigned to the same slot. This made the schedule easier to manage and more realistic.
- **Shift Completion Logic Functional**
  People were automatically marked as “Finished” once they’d taken their breaks and the timing rules allowed it. This reduced clutter and made it clear who was still active.

## 7.2 Technical Performance

**From a backend and system integration standpoint, the project demonstrated:**

- **High-Volume Data Handling**
  I ran test days with hundreds of staff, and the system kept up by splitting the data into batches (first 100, next 200, etc.). The backend processed everything smoothly and combined results into one clean schedule.
- **Reliable Prompt-Driven Scheduling**
  The GPT output followed the right format — {ID} {FirstBreak} {SecondBreak} — and made smart decisions using the input constraints. There was no need to "fix" the output; it just worked once the prompt was tuned properly.
- **Clock-Based Automation**
  The test clock drove everything: break alerts, transitions to On Break, and automatic finishes. This let me simulate an entire day's schedule in minutes without having to manually trigger anything.
- **Error Resilience and Validation**
  If a person had a malformed or missing time entry, the system skipped them and logged a warning. This made testing way smoother and prevented small errors from killing the whole run.

---

## 7.3 Feedback and Use Case Validation

**Even though the system wasn't deployed in a live airport, I tested it against real shift data and walk-through scenarios from my time on the job. I showed it to ASU officers, coordinators, and duty managers — and they recognized the logic straight away.**

**Most importantly, it solved problems they deal with daily: break tracking, role coverage, and overload during peak times. The AI didn't just guess — it felt like it was working with them, not against them. That told me I was on the right track.**

---

# 8. Future Work

**I see this project as more than just a final-year system — it's something that could grow into a real scheduling platform for fast-moving environments like airports. The current version already handles break planning and automation, but there are a lot of exciting directions it could go next. Here's what I'd work on if I had more time (or if this moves into production).**

## 8.1 Expansion to Airline Operations

**After showing the system to some people in Ryanair's HR team, we started discussing how this could be adapted for flight crews — pilots, cabin staff, and ground teams. That's a whole different set of rules:**

- Flight hours and rest periods under CAA and EASA regulations
- Scheduling across multiple airports with different staffing levels and time zones
- Roles like flight deck, cabin crew, or dispatch that all have unique timing and coverage needs

**To support that, I'd need to plug into Ryanair's HR databases and flight schedules, and rewrite parts of the logic to fit aviation-specific duty rules. But the core of the system — shift tracking, break logic, and AI suggestions — would still apply.**

---

## 8.2 Predictive Scheduling and Demand Forecasting

**Right now, the system reacts to a set schedule. But in the future, it could go one step further: predicting how many staff are needed before a shift is even planned.**

**Using historic traffic patterns, scheduled flights, and queue data, the system could say:**
"Tomorrow looks like a light day until 10AM — you only need 95 staff instead of 120."

**That would save money, reduce overstaffing, and help hit queue targets more reliably. I'd star with a basic forecasting model that factors in:**

- Day of week / season
- Flight schedules
- Terminal-specific traffic history
- Past queue performance

**It would feed directly into roster planning so that coordinators could plan smarter, not just react faster.**

---

## 8.3 Role Coverage and Machine Awareness

**While the current version makes sure there's enough staff on duty and breaks are spaced well, it doesn't yet track which specific machines (X-ray, scanner, etc.) are covered. That's something I'd like to build in.**

**Future updates could include:**

- A visual map of which machines are manned at any given moment

- Smart role handovers when someone goes on break (e.g., “James is leaving WTMD at 07:00, so Sarah will take over”)
- Warnings if any priority role is left uncovered

**This would let the AI go from just balancing headcount to managing the actual functionality of the floor.**

---

### 8.4 Enhanced Supervisor Interface

**Right now, the interface is built with coordinators in mind, but a real airport rollout would need a more powerful dashboard for supervisors.**

**I’d like to build a live panel with:**

- Drag-and-drop rescheduling
- Editable break plans with override options
- Real-time stats (coverage, breaks overdue, etc.)

**Think of it like a mission control — the AI suggests a plan, and the supervisor can either go with it or tweak things based on what’s actually happening.**

---

### 8.5 Formal Evaluation and Deployment

**The next big step would be to run a field test — maybe during a quieter period like late evening or shoulder season.**

**That trial could log things like:**

- Queue times vs. predicted traffic
- Whether break coverage held up
- How often supervisors needed to step in

**From there, I’d collect feedback from frontline staff and shift leads, then refine the AI logic even further. With enough proof, the system could be certified and scaled to other terminals — or even other industries that rely on shift-based roles.**

---

## 9. Acknowledgements

**I’d like to thank everyone who helped shape this project, both directly and indirectly.**

**Working with the Dublin Airport Authority (DAA) for the past two years as an Airport Search Unit Officer gave me a unique perspective into how things actually operate on the ground. From chatting with supervisors, coordinators, and duty managers to**

**observing how shifts are handled day-to-day, this experience really guided the direction and priorities of the system. Their feedback was a big part of why the project works the way it does.**

**I also want to thank the Ryanair HR team, with whom I've been in touch recently about adapting the system for their own teams — including cabin crew and pilots. Their interest and input have helped highlight the flexibility and wider potential of what I've built.**

**A big thanks also goes to the lecturers and staff in the Computer Science department. The knowledge I gained over the last few years—especially in React, Django, and system integration—made this project possible. I'd also like to acknowledge my internship at Davy Stockbrokers, where I learned how to tackle large, messy data problems and build structured solutions from the ground up. That experience proved to be incredibly useful when developing the AI components of the project.**

**Finally, thanks to anyone who gave feedback, helped test the system, or even just listened while I worked through the logic out loud.**