

Challenge Project: Staff Attendance Record System

Project Overview

Build a comprehensive **Staff Attendance Record REST API** using Poetry for project management and FastAPI for the backend. This system manages employee information, tracks daily attendance (check-in/check-out), calculates working hours, and generates attendance reports.

Project Goal

Create a RESTful API that enables HR departments to manage staff attendance records, monitor punctuality, track leave requests, and generate attendance reports for payroll processing.

Technical Requirements

1. Poetry Project Setup (25% of grade)

Required Setup:

- Initialize project: `poetry new attendance-system`
- Configure `pyproject.toml`:
 - Project name: "attendance-system"
 - Description: "Staff Attendance Record Management API"
 - Python version: `^3.10`
 - Authors information

Required Dependencies:

- **Runtime:**
 - `fastapi` - Web framework
 - `uvicorn[standard]` - ASGI server
 - `pydantic` - Data validation
 - `python-dateutil` - Date/time utilities
 - `python-multipart` - Form data handling
- **Development:**
 - `pytest` - Testing framework
 - `black` - Code formatter
 - `isort` - Import organizer

Required Operations:

- Create and activate Poetry virtual environment
- Install all dependencies
- Generate `poetry.lock` file
- Export to `requirements.txt`
- Run app using `poetry run uvicorn attendance_system.main:app --reload`

Deliverables:

- Complete `pyproject.toml` with all dependencies
- Screenshot of poetry env info
- Screenshot of poetry show

2. FastAPI Application Development (75% of grade)

Project Structure

```
text
attendance-system/
└── attendance_system/
    ├── __init__.py
    ├── main.py      # FastAPI app initialization
    ├── models.py    # Pydantic models
    ├── routes/
    │   ├── __init__.py
    │   ├── employees.py  # Employee management
    │   ├── attendance.py # Attendance records
    │   └── reports.py   # Reports and analytics
    ├── database.py   # In-memory data storage
    ├── utils.py      # Helper functions
    └── enums.py      # Enum definitions
    └── tests/
        └── test_attendance.py
    └── pyproject.toml
    └── README.md
```

Data Models

Department Enum

```
python
class Department(str, Enum):
    HR = "HR"
    IT = "IT"
    FINANCE = "Finance"
    OPERATIONS = "Operations"
    SALES = "Sales"
    MARKETING = "Marketing"
```

Employee Model

```
python
class Employee(BaseModel):
    id: int
    employee_code: str      # unique, format: EMP001
    full_name: str          # max 100 chars
    email: str              # valid email format
    department: Department
    position: str           # max 50 chars
    hire_date: date
    is_active: bool         # default: True
    created_at: datetime
```

AttendanceStatus Enum

```
python
class AttendanceStatus(str, Enum):
    PRESENT = "Present"
    LATE = "Late"        # check-in after 9:15 AM
    HALF_DAY = "Half Day"  # < 4 hours worked
    ABSENT = "Absent"
    ON_LEAVE = "On Leave"
```

AttendanceRecord Model

```
python
class AttendanceRecord(BaseModel):
    id: int
    employee_id: int
    date: date
    check_in_time: Optional[datetime]
    check_out_time: Optional[datetime]
    status: AttendanceStatus
    working_hours: Optional[float]  # auto-calculated
    notes: Optional[str]            # max 200 chars
    created_at: datetime
    updated_at: datetime
```

LeaveType Enum

```
python
class LeaveType(str, Enum):
    SICK_LEAVE = "Sick Leave"
    ANNUAL_LEAVE = "Annual Leave"
    PERSONAL_LEAVE = "Personal Leave"
    UNPAID_LEAVE = "Unpaid Leave"
    MATERNITY_LEAVE = "Maternity Leave"
```

LeaveRequest Model

```
python
class LeaveRequest(BaseModel):
    id: int
```

```
employee_id: int
leave_type: LeaveType
start_date: date
end_date: date
total_days: int      # auto-calculated
reason: str          # max 300 chars
status: str          # Pending/Approved/Rejected
requested_at: datetime
approved_by: Optional[int]  # employee_id of approver
approved_at: Optional[datetime]
```

Required API Endpoints

Employee Management (20 points)

1. **POST /employees**
 - o Create new employee
 - o Validate employee_code uniqueness
 - o Validate email format
 - o Return 201 Created
2. **GET /employees**
 - o List all employees
 - o Filter by:
 - department (query param)
 - is_active (query param, default: True)
 - o Support pagination: skip and limit
3. **GET /employees/{employee_id}**
 - o Get employee details
 - o Return 404 if not found
4. **GET /employees/code/{employee_code}**
 - o Search employee by employee code
 - o Return 404 if not found
5. **PUT /employees/{employee_id}**
 - o Update employee information
 - o Cannot change employee_code
6. **PATCH /employees/{employee_id}/deactivate**
 - o Set is_active = False
 - o Return updated employee
7. **DELETE /employees/{employee_id}**
 - o Delete employee (soft delete by setting is_active = False)
 - o Return 204 No Content

Attendance Records (30 points)

1. **POST /attendance/check-in**
 - o Record employee check-in
 - o Request body: employee_id, check_in_time (optional, default: now)

- Validate: One check-in per employee per day
 - Auto-determine status (Late if after 9:15 AM)
 - Return 201 Created
2. **PATCH /attendance/check-out**
 - Record employee check-out
 - Request body: `employee_id, check_out_time` (optional, default: now)
 - Calculate working hours automatically
 - Update status (Half Day if < 4 hours)
 - Return updated record
 3. **GET /attendance**
 - List all attendance records
 - Filter by:
 - `employee_id` (query param)
 - `date` (query param, format: YYYY-MM-DD)
 - `date_from` and `date_to` (date range)
 - `status` (query param)
 - `department` (query param)
 - Support pagination
 4. **GET /attendance/{attendance_id}**
 - Get specific attendance record
 - Return 404 if not found
 5. **GET /attendance/employee/{employee_id}/today**
 - Get today's attendance for specific employee
 - Return 404 if no record
 6. **GET /attendance/employee/{employee_id}/month**
 - Get current month's attendance for employee
 - Optional: `year` and `month` query params
 7. **PUT /attendance/{attendance_id}**
 - Update attendance record (for corrections)
 - Recalculate working hours if times changed
 8. **POST /attendance/mark-absent**
 - Mark employee as absent for specific date
 - Request body: `employee_id, date`
 - Create record with status = ABSENT
 9. **DELETE /attendance/{attendance_id}**
 - Delete attendance record
 - Return 204 No Content

Leave Management (15 points)

1. **POST /leaves**
 - Create leave request
 - Validate: `start_date <= end_date`
 - Auto-calculate `total_days` (excluding weekends)
 - Status defaults to "Pending"
 - Return 201 Created
2. **GET /leaves**

- List all leave requests
 - Filter by:
 - employee_id (query param)
 - status (query param)
 - leave_type (query param)
 - Support pagination
3. **GET /leaves/{leave_id}**
- Get specific leave request
 - Return 404 if not found
4. **PATCH /leaves/{leave_id}/approve**
- Approve leave request
 - Request body: approved_by (employee_id)
 - Update status to "Approved"
 - Set approved_at timestamp
 - Auto-create attendance records with ON_LEAVE status
5. **PATCH /leaves/{leave_id}/reject**
- Reject leave request
 - Update status to "Rejected"
6. **DELETE /leaves/{leave_id}**
- Cancel leave request (if Pending)
 - Return 409 if already approved/rejected
-

Reports & Analytics (20 points)

1. **GET /reports/daily-summary**
- Query params: date (default: today)
 - Return:
 - Total employees
 - Present count
 - Absent count
 - Late count
 - On leave count
 - List of absent employees
 - List of late employees
2. **GET /reports/employee/{employee_id}/monthly-summary**
- Query params: year, month (default: current month)
 - Return:
 - Total working days
 - Days present
 - Days late
 - Days absent
 - Days on leave
 - Total working hours
 - Average check-in time
 - Attendance percentage
3. **GET /reports/department/{department}/attendance**
- Query params: date_from, date_to

- Return department-wide statistics:
 - Total employees in department
 - Average attendance rate
 - Total late arrivals
 - Total absences
 - 4. **GET /reports/monthly-attendance**
 - Query params: year, month, department (optional)
 - Return CSV-format data of all attendance records
 - Include: employee_code, name, date, check-in, check-out, hours, status
 - 5. **GET /reports/punctuality-ranking**
 - Query params: date_from, date_to
 - Return employees ranked by:
 - Attendance rate (descending)
 - Number of late arrivals (ascending)
 - Limit to top 10
-

Advanced FastAPI Features Required

Validation & Business Logic (10 points)

- **Employee Code Format:** Must match pattern `EMP\d{3}` (e.g., EMP001)
- **Email Validation:** Use Pydantic EmailStr
- **Working Hours Calculation:**
 - Auto-calculate from check-in and check-out times
 - Round to 2 decimal places
 - Subtract 1 hour for lunch break if > 6 hours
- **Late Detection:** Auto-mark as Late if check-in after 9:15 AM
- **Half-Day Detection:** Auto-mark as Half Day if working hours < 4
- **Weekend Detection:** Cannot mark attendance on weekends (return 400 error)
- **Leave Days Calculation:** Exclude weekends when calculating total_days
- **One Record Per Day:** Prevent duplicate attendance records for same employee on same date

Error Handling (5 points)

- **400 Bad Request:** Invalid date, weekend attendance, duplicate record
- **404 Not Found:** Employee, attendance record, or leave request not found
- **409 Conflict:** Duplicate employee code, already checked in/out
- **422 Unprocessable Entity:** Validation errors with detailed messages

Dependencies (5 points)

- Create dependency function for database access
- Create dependency for employee validation (check if exists and active)
- Create dependency for date validation (not future date, not weekend)
- Use `Depends()` in endpoint parameters

Background Tasks (5 points)

- Log all check-in/check-out to file (timestamp, employee_code, action)
- Send notification when leave is approved (print to console)

CORS & Middleware (5 points)

- Enable CORS for all origins
- Add custom middleware to log request processing time
- Add middleware to add custom header: `X-Attendance-API-Version: 1.0`

API Documentation (5 points)

- Add app description and version in FastAPI initialization
 - Use route tags: "Employees", "Attendance", "Leaves", "Reports"
 - Add docstrings with examples for all endpoints
 - Provide request/response examples in schema
-

Bonus Challenges (+25% extra credit)

1. **Overtime Tracking** (+5%):
 - Add overtime hours calculation (hours worked > 8)
 - Endpoint: `GET /reports/overtime`
 2. **Export Reports** (+5%):
 - Generate CSV export for monthly reports
 - Endpoint: `GET /reports/export/csv`
 3. **Bulk Check-in** (+5%):
 - Upload CSV file with multiple check-ins
 - Endpoint: `POST /attendance/bulk-check-in`
 4. **Attendance Alerts** (+5%):
 - Endpoint to get employees with < 80% attendance
 - Query param: date range
 5. **Testing Suite** (+10%):
 - Write pytest tests for all major endpoints
 - Test validation rules
 - Test business logic (working hours calculation)
 - 70%+ code coverage
-

Evaluation Rubric

Poetry Setup (25 points)

- Project initialization: 5 points

- Dependency management: 10 points
- Virtual environment setup: 5 points
- Documentation: 5 points

FastAPI Development (75 points)

Project Structure (5 points)

- Organized file layout with separate routers

Data Models (10 points)

- Proper Pydantic models with validation
- Correct use of Enums
- Field constraints and validators

Employee Endpoints (20 points)

- All 7 endpoints implemented correctly
- Proper validation and error handling

Attendance Endpoints (30 points)

- All 9 endpoints working
- Correct business logic (late detection, hours calculation)
- One record per day validation

Leave Endpoints (15 points)

- All 6 endpoints functional
- Correct date range validation
- Auto-create attendance records on approval

Reports Endpoints (20 points)

- All 5 report endpoints with accurate calculations
- Proper aggregation and filtering

Advanced Features (10 points)

- Validation, dependencies, background tasks

Code Quality (10 points)

- Clean, readable code
 - Proper error handling
 - Good variable names
-

Submission Requirements

1. Project Package (ZIP)

Include:

- Complete source code
- `pyproject.toml` and `poetry.lock`
- `README.md` with:
 - Setup instructions
 - How to run with Poetry and Uvicorn
 - API endpoint documentation
 - Example requests for each endpoint
 - Business rules explanation

2. Documentation (PDF)

Include:

- **Setup Screenshots:**
 - Poetry environment info
 - Dependency tree (`poetry show --tree`)
 - Running application in terminal
- **Testing Screenshots** (at least 10):
 - Interactive docs at `/docs`
 - Creating employees
 - Check-in/check-out flow
 - Leave request and approval
 - Daily summary report
 - Monthly attendance report
 - Error responses (404, 400, 409)
- **Code Highlights:**
 - Pydantic models
 - Working hours calculation logic
 - One complex endpoint implementation

3. Demo (15-20 minutes)

Demonstrate:

1. Project setup with Poetry (2 min)
2. Running application (1 min)
3. Complete workflow (10 min):
 - Create employees from different departments
 - Check-in employees (on-time and late)
 - Check-out employees
 - Mark someone absent
 - Create and approve leave request

- Generate daily summary
 - Generate monthly employee report
4. Error handling examples (3 min)
 5. Code walkthrough (4 min)
-

Testing Workflow for Students

Day 1 Workflow:

1. Create 5 employees from different departments
2. Check-in 3 employees (2 on-time, 1 late)
3. Mark 1 employee absent
4. Create leave request for 1 employee
5. Generate daily summary

Day 2 Workflow:

1. Check-out all checked-in employees
2. Verify working hours calculation
3. Approve leave request
4. Verify attendance records created for leave period
5. Generate monthly report for one employee

Edge Cases to Test:

- Cannot check-in twice on same day
 - Cannot check-out without check-in
 - Cannot mark attendance on weekend
 - Late status auto-assigned after 9:15 AM
 - Half-day status for < 4 hours
 - Working hours calculation with lunch break
 - Leave days calculation excludes weekends
 - Cannot delete approved leave
 - Employee code must be unique
 - Email must be valid format
-

Success Criteria

Students successfully complete the project when they:

- Set up Poetry project with all dependencies
- Create well-structured FastAPI application with multiple routers
- Implement all required endpoints with correct HTTP methods

- Apply proper Pydantic validation with custom rules
- Implement business logic (working hours, late detection, etc.)
- Handle errors appropriately with meaningful messages
- Generate accurate reports and analytics
- Deploy and run application with Uvicorn
- Provide comprehensive documentation
- Demonstrate complete workflow in video