Smart Meeting Room Management System

Detailed Deliverables & Commit Plan

Software Tools Lab - Fall 2025-2026

Team Members: Yateem & Fouani

November 21, 2025

Contents

1 Project Overview & Division Strategy

1.1 Project Structure

The project name will be: `smartmeetingroom_Yateem_Fouani`

1.2 Technology Stack Based on Lab Materials

- **Backend Framework**: Flask (Lab 1)
- **Database**: SQLite with SQLAlchemy ORM (Lab 2)
- **Testing**: Pytest (Lab 7)
- **Documentation**: Sphinx with Docstrings (Lab 3)
- **API Testing**: Postman Collections (Lab 5)
- **Build Automation**: PyBuilder & Makefile (Lab 6)
- **Containerization**: Docker & Docker Compose (Lab 8)
- **Profiling**: cProfile, memory_profiler, coverage.py (Lab 9)
- **Version Control**: Git with GitHub (Lab 4)
- **Configuration**: python-dotenv, ConfigParser (Lab 11)
- **CI/CD**: Jenkins pipeline (Lab 10)

1.3 Service Division

Service	Owner	Port
Users Service	Yateem	5001
Rooms Service	Fouani	5002
Bookings Service	Yateem	5003
Reviews Service	Fouani	5004
Database Container	Shared	5432

1.4 Part II Tasks Division

Team Member	Part II Tasks (2 each)
Yateem	1. Circuit Breaker Pattern (Inter-Service Communication) 2. Asynchronous Messaging with RabbitMQ
Fouani	1. Caching with Redis 2. Real-Time Dashboard with Grafana

2 Detailed Commit Plan (35 Total Commits)

2.1 Phase 1: Project Setup & Configuration (Commits 1-5)

#	Owner	Commit Description & Detailed Tasks
1	Yateem	**Initial project setup and structure**

- Create main project folder: `smartmeetingroom_Yateem_Fouani`
- Initialize Git repository with `.gitignore` for Python
- Create directory structure:

```
smartmeetingroom_Yateem_Fouani/
          services/
                users/
                rooms/
                bookings/
                reviews/
          database/
          tests/
          docs/
          postman/
          docker/
          configs/
```

- Create `requirements.txt` with initial dependencies:

```
Flask==2.3.2
SQLAlchemy==2.0.15
pytest==7.3.1
sphinx==7.0.1
python-dotenv==1.0.0
flask-cors==4.0.0
flask-jwt-extended==4.5.2
```

- Create `README.md` with project overview
- Push to GitHub with detailed commit message

#	Owner	Commit Description & Detailed Tasks

2	Fouani	**Database infrastructure and Rooms/Reviews models**
		• Create `database/database.py` for connection handling
		• Create `database/init_db.py` for initialization
		• Create migration scripts folder
		• Create `database/models.py` with Room and Review SQLAlchemy models:
		`# Room model with fields: id, name, capacity, equipment,`
		`# location, floor, building, amenities, status, hourly_rate,`
		`# image_url, created_at, updated_at`
		`# Review model with fields: id, user_id, room_id, booking_id,`
		`# rating, title, comment, pros, cons, is_flagged, flag_reason,`
		`# flagged_by, flagged_at, is_hidden, hidden_reason,`
		`# helpful_count, unhelpful_count, created_at, updated_at, edited_at`
		• Add indexes for Room and Review tables
		• Create base model class with common fields

#	Owner	Commit Description & Detailed Tasks

3	Yateem	**Environment configuration, security setup, and Users/Bookings models**
		• Create `configs/.env.template` with all environment variables:
		`DATABASE_URL=sqlite:///smartmeetingroom.db`
		`JWT_SECRET_KEY=your-secret-key-here`
		`FLASK_ENV=development`
		`USER_SERVICE_PORT=5001`
		`ROOM_SERVICE_PORT=5002`
		`BOOKING_SERVICE_PORT=5003`
		`REVIEW_SERVICE_PORT=5004`
		• Create `configs/config.py` with ConfigParser implementation
		• Setup `utils/auth.py` for JWT authentication
		• Create `utils/validators.py` for input validation
		• Setup `utils/sanitizers.py` for SQL injection prevention
		• Create password hashing utilities using bcrypt
		• Add to `database/models.py` User and Booking SQLAlchemy models:
		`# User model with fields: id, username, password_hash,`
		`# email, full_name, role, is_active, created_at, updated_at,`
		`# last_login, failed_login_attempts, locked_until`
		`# Booking model with fields: id, user_id, room_id, title,`
		`# description, start_time, end_time, status, attendees,`
		`# is_recurring, recurrence_pattern, recurrence_end_date,`
		`# cancellation_reason, cancelled_at, cancelled_by,`
		`# created_at, updated_at`
		• Add relationship mappings between all models (User-Booking, Booking-Room, User-Review, etc.)
		• Create constraints for booking time validation and overlap prevention
		• Add indexes for User and Booking tables
		• Create Audit Log model for tracking all system changes

#	Owner	Commit Description & Detailed Tasks

4	Fouani	**Docker environment and base images**
		• Create `docker/Dockerfile.base` for shared Python base image:
		`FROM python:3.10-slim`
		`WORKDIR /app`
		`COPY requirements.txt .`
		`RUN pip install --no-cache-dir -r`
		`   requirements.txt`
		• Create individual Dockerfiles for each service
		• Create `docker-compose.yml` with all services
		• Setup Docker network for inter-service communication
		• Create `docker/database/Dockerfile` for PostgreSQL
		• Add volume mappings for persistent data

5	Yateem	**Logging, error handling, and base utilities**
		• Create `utils/logger.py` with structured logging:
		`import logging`
		`logging.basicConfig(`
		`    format='%(asctime)s␣-␣%(name)s␣-␣%(`
		`       levelname)s␣-␣%(message)s',`
		`    level=logging.INFO,`
		`    handlers=[`
		`        logging.FileHandler('app.log'),`
		`        logging.StreamHandler()`
		`    ]`
		`)`
		• Setup `utils/exceptions.py` with custom exceptions
		• Create `utils/decorators.py` with rate limiting decorator
		• Setup `utils/responses.py` for standardized API responses
		• Create audit logging functionality

#	Owner	Commit Description & Detailed Tasks

6	Yateem	**Users Service - Core authentication APIs**
		• Create `services/users/app.py` with Flask app
		• Implement `/api/auth/register` endpoint:
		`@app.route('/api/auth/register', methods=['POST'])`
		`def register():`
		`    # Validate input (name, username, password, email, role)`
		`    # Check if username/email exists`
		`    # Hash password with bcrypt`
		`    # Create user in database`
		`    # Return JWT token and user info`
		• Implement `/api/auth/login` endpoint
		• Add input validation and sanitization
		• Create JWT token generation logic
		• Add password strength validation

7	Fouani	**Rooms Service - CRUD operations**
		• Create `services/rooms/app.py` with Flask app
		• Implement `POST /api/rooms` - Add new room:
		`@app.route('/api/rooms', methods=['POST'])`
		`@jwt_required()`
		`@admin_required`
		`def add_room():`
		`    # Validate room data (name, capacity, equipment, location)`
		`    # Check for duplicate room names`
		`    # Create room in database`
		`    # Return room details with ID`
		• Implement `GET /api/rooms` - List all rooms
		• Implement `GET /api/rooms/<id>` - Get room by ID
		• Add filtering by capacity, location, equipment
		• Implement pagination for room listings

#	Owner	Commit Description & Detailed Tasks

8	Yateem	**Users Service - Profile management**
		• Implement `PUT /api/users/profile` - Update profile:
		`@app.route('/api/users/profile',`
		`    methods=['PUT'])`
		`@jwt_required()`
		`def update_profile():`
		`    # Get current user from JWT`
		`    # Validate updated fields`
		`    # Update user in database`
		`    # Return updated user info`
		• Implement `DELETE /api/users/<id>` - Delete user (admin only)
		• Implement `GET /api/users` - Get all users (admin only)
		• Implement `GET /api/users/<username>` - Get user by username
		• Add role-based access control (RBAC) middleware

9	Fouani	**Rooms Service - Advanced room management**
		• Implement `PUT /api/rooms/<id>` - Update room:
		`@app.route('/api/rooms/<int:room_id>',`
		`    methods=['PUT'])`
		`@jwt_required()`
		`@admin_required`
		`def update_room(room_id):`
		`    # Validate room exists`
		`    # Check for booking conflicts if capacity reduced`
		`    # Update room details`
		`    # Log changes for audit`
		• Implement `DELETE /api/rooms/<id>` - Delete room
		• Implement `GET /api/rooms/available` - Check availability
		• Add room status management (available/maintenance/booked)
		• Create equipment inventory tracking

#	Owner	Commit Description & Detailed Tasks

10	Yateem	**Bookings Service - Core booking operations**
		• Create `services/bookings/app.py` with Flask app
		• Implement `POST /api/bookings` - Create booking:
		`@app.route('/api/bookings', methods=['POST'])`
		`@jwt_required()`
		`def create_booking():`
		`    # Validate booking data (room_id, start_time, end_time)`
		`    # Check room availability for time slot`
		`    # Check for conflicting bookings`
		`    # Create booking record`
		`    # Send confirmation (prepare for async messaging)`
		• Implement conflict detection algorithm
		• Add booking validation rules (min/max duration)
		• Create recurring booking support

11	Fouani	**Reviews Service - Review submission**
		• Create `services/reviews/app.py` with Flask app
		• Implement `POST /api/reviews` - Submit review:
		`@app.route('/api/reviews', methods=['POST'])`
		`@jwt_required()`
		`def submit_review():`
		`    # Validate user has completed booking for room`
		`    # Validate rating (1-5) and comment`
		`    # Sanitize comment for XSS/SQL injection`
		`    # Check for duplicate reviews`
		`    # Store review in database`
		• Implement profanity filter for comments
		• Add sentiment analysis preparation
		• Create review validation rules

#	Owner	Commit Description & Detailed Tasks

12	Yateem	**Bookings Service - Management endpoints**
		• Implement `GET /api/bookings` - View all bookings:
		`@app.route('/api/bookings', methods=['GET'])`
		`@jwt_required()`
		`def get_bookings():`
		`    # Check user role for filtering`
		`    # Admin sees all, users see own bookings`
		`    # Add date range filtering`
		`    # Include room and user details`
		`    # Implement pagination`
		• Implement `PUT /api/bookings/<id>` - Update booking
		• Implement `DELETE /api/bookings/<id>` - Cancel booking
		• Add cancellation policy logic
		• Create booking status workflow

13	Fouani	**Reviews Service - Review management**
		• Implement `PUT /api/reviews/<id>` - Update review:
		`@app.route('/api/reviews/<int:review_id>', methods=['PUT'])`
		`@jwt_required()`
		`def update_review(review_id):`
		`    # Verify review ownership`
		`    # Validate updated content`
		`    # Update review with edit history`
		`    # Trigger re-moderation if needed`
		• Implement `DELETE /api/reviews/<id>` - Delete review
		• Implement `GET /api/reviews/room/<id>` - Get room reviews
		• Add review aggregation statistics
		• Create helpful/unhelpful voting system

#	Owner	Commit Description & Detailed Tasks

14	Yateem	**Users Service - Booking history integration**
		• Implement `GET /api/users/<id>/bookings` - User booking history:
		`@app.route('/api/users/<int:user_id>/bookings', methods=['GET'])`
		`@jwt_required()`
		`def get_user_bookings(user_id):`
		`    # Verify access rights (own data or admin)`
		`    # Call Bookings Service API`
		`    # Format response with room details`
		`    # Add statistics (total bookings, favorite rooms)`
		• Create inter-service communication helper
		• Add request retry logic
		• Implement response caching preparation
		• Create user activity dashboard data

15	Fouani	**Reviews Service - Moderation features**
		• Implement `POST /api/reviews/<id>/flag` - Flag review:
		`@app.route('/api/reviews/<int:review_id>/flag', methods=['POST'])`
		`@jwt_required()`
		`def flag_review(review_id):`
		`    # Record who flagged and reason`
		`    # Increment flag counter`
		`    # Auto-hide if threshold reached`
		`    # Notify moderators`
		• Implement `GET /api/reviews/flagged` - Get flagged reviews (moderator)
		• Implement `PUT /api/reviews/<id>/moderate` - Moderate review
		• Create moderation queue system
		• Add moderation audit log

#	Owner	Commit Description & Detailed Tasks

16	Yateem	**Bookings Service - Availability checker**
		• Implement `POST /api/bookings/check-availability`:
		`@app.route('/api/bookings/check-availability', methods=['POST'])`
		`def check_availability():`
		`    # Parse date range and room requirements`
		`    # Query existing bookings`
		`    # Calculate available time slots`
		`    # Return availability matrix`
		`    # Include suggested alternatives`
		• Create availability calendar generator
		• Add conflict resolution suggestions
		• Implement smart scheduling algorithm
		• Create booking optimization logic

17	Fouani	**Postman collection for Rooms & Reviews services**
		• Create `postman/Rooms_Service.postman_collection.json`:
		`{`
		`  "info": {`
		`    "name": "Rooms Service API",`
		`    "description": "Complete API documentation for Rooms"`
		`  },`
		`  "item": [`
		`    // Add all room endpoints with examples`
		`    // Include test scripts for each endpoint`
		`    // Add environment variables`
		`    // Create request chaining`
		`  ]`
		`}`
		• Create Reviews Service collection with all endpoints
		• Add pre-request scripts for authentication
		• Create test scenarios with assertions
		• Add example responses and error cases

#	Owner	Commit Description & Detailed Tasks
18	Yateem	**Postman collection for Users & Bookings services** • Create `postman/Users_Service.postman_collection.json` • Create `postman/Bookings_Service.postman_collection.json` • Add environment file with variables: `{ "name": "Development", "values": [ {"key": "base_url", "value": "http://localhost"}, {"key": "jwt_token", "value": ""}, {"key": "user_id", "value": ""} ] }` • Create end-to-end test flows • Add data generation scripts
19	Fouani	**Unit tests for Rooms & Reviews services** • Create `tests/test_rooms_service.py`: `import pytest` `from services.rooms.app import app` `@pytest.fixture` `def client():` `app.config['TESTING'] = True` `with app.test_client() as client:` `yield client` `def test_add_room(client, auth_headers):` `response = client.post('/api/rooms', json={'name': 'Conference A', 'capacity': 10}, headers=auth_headers)` `assert response.status_code == 201` • Create comprehensive test suite for Reviews • Add fixture for test database • Create mock data generators • Implement edge case testing

Code in row 18:

```json
{
  "name": "Development",
  "values": [
    {"key": "base_url", "value": "http://localhost"},
    {"key": "jwt_token", "value": ""},
    {"key": "user_id", "value": ""}
  ]
}
```

Code in row 19:

```python
import pytest
from services.rooms.app import app

@pytest.fixture
def client():
    app.config['TESTING'] = True
    with app.test_client() as client:
        yield client

def test_add_room(client, auth_headers):
    response = client.post('/api/rooms',
        json={'name': 'Conference A', 'capacity': 10},
        headers=auth_headers)
    assert response.status_code == 201
```

#	Owner	Commit Description & Detailed Tasks

20	Yateem	**Unit tests for Users & Bookings services**
		• Create `tests/test_users_service.py`
		• Create `tests/test_bookings_service.py`
		• Add integration tests between services:
		`def test_booking_workflow(client):`
		`    # Register user`
		`    # Login and get token`
		`    # Create booking`
		`    # Verify booking in history`
		`    # Cancel booking`
		`    # Verify cancellation`
		• Create performance test scenarios
		• Add security testing (SQL injection, XSS)

2.2 Phase 3: Documentation & Profiling (Commits 21-25)

#	Owner	Commit Description & Detailed Tasks
21	Fouani	**Sphinx documentation setup and API docs**

- Initialize Sphinx documentation:

```
cd docs/
sphinx-quickstart
# Configure conf.py with autodoc extensions
# Set up RTD theme
```

- Create comprehensive docstrings for all functions:

```
def create_booking(room_id: int, start_time: datetime,
                      end_time: datetime)
                          -> dict:
    """
    Create a new booking for a room.

    Args:
        room_id: The ID of the room to book
        start_time: Booking start time
        end_time: Booking end time

    Returns:
        dict: Booking confirmation with details

    Raises:
        ConflictError: If time slot is already booked
        ValidationError: If input data is invalid
    """
```

- Generate HTML documentation
- Create API reference guide

#	Owner	Commit Description & Detailed Tasks

22	Yateem	**Performance profiling implementation**
		• Create `profiling/performance_tests.py`:
		`import cProfile`
		`import pstats`
		`from memory_profiler import profile`
		
		`@profile`
		`def test_booking_performance():`
		`    # Create 1000 bookings`
		`    # Measure time and memory`
		`    # Generate performance report`
		
		`cProfile.run('test_booking_performance()', 'profile_stats')`
		`stats = pstats.Stats('profile_stats')`
		`stats.sort_stats('cumulative')`
		`stats.print_stats(20)`
		• Run memory profiling on all services
		• Create load testing scripts
		• Generate performance reports with graphs
		• Identify and document bottlenecks

#	Owner	Commit Description & Detailed Tasks

23	Fouani	**Code coverage and test reporting**
		• Setup coverage.py configuration:
		`# .coveragerc`
		`[run]`
		`source = services`
		`omit = */tests/*,*/venv/*`
		
		`[report]`
		`precision = 2`
		`show_missing = True`
		`skip_covered = False`
		
		`[html]`
		`directory = coverage_html_report`
		• Run coverage analysis:
		`coverage run -m pytest tests/`
		`coverage report -m`
		`coverage html`
		• Generate coverage badges
		• Create test report documentation
		• Document untested edge cases

#	Owner	Commit Description & Detailed Tasks
24	Yateem	**Docker finalization and orchestration** • Finalize `docker-compose.yml`:

```
version: '3.8'
services:
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: smartmeetingroom
      POSTGRES_USER: admin
      POSTGRES_PASSWORD:
         secure_password
    volumes:
      - postgres_data:/var/lib/
         postgresql/data
    ports:
      - "5432:5432"

  users-service:
    build:
      context: .
      dockerfile: docker/Dockerfile.
         users
    ports:
      - "5001:5001"
    depends_on:
      - postgres
    environment:
      DATABASE_URL: postgresql://admin:
         secure_password@postgres:5432/
         smartmeetingroom
```

- Create health check endpoints
- Add container restart policies
- Setup logging volumes

#	Owner	Commit Description & Detailed Tasks
25	Fouani	**Makefile and build automation** • Create comprehensive `Makefile`:

```
.PHONY: install test build run clean
   docs

install:
        python -m venv venv
        . venv/bin/activate && pip
           install -r requirements.txt

test:
        . venv/bin/activate && pytest
           tests/ -v --cov=services

build:
        docker-compose build

run:
        docker-compose up -d

clean:
        docker-compose down
        find . -type d -name
           __pycache__ -exec rm -rf {}
           +
        rm -rf coverage_html_report

docs:
        cd docs && make html

profile:
        python profiling/
           performance_tests.py

all: install test build docs
```

- Add PyBuilder configuration
- Create CI/CD preparation scripts

2.3 Phase 4: Part II Enhancements (Commits 26-33)

#	Owner	Commit Description & Detailed Tasks
#	Owner	Commit Description & Detailed Tasks

26	Yateem	**Circuit Breaker Pattern implementation**
		• Create `utils/circuit_breaker.py`:
		`import time`
		`from functools import wraps`
		
		`class CircuitBreaker:`
		`    def __init__(self,`
		`       failure_threshold=5,`
		`                 recovery_timeout=60,`
		`                 expected_exception=`
		`                    Exception):`
		`        self.failure_threshold =`
		`           failure_threshold`
		`        self.recovery_timeout =`
		`           recovery_timeout`
		`        self.expected_exception =`
		`           expected_exception`
		`        self.failure_count = 0`
		`        self.last_failure_time = None`
		`        self.state = 'CLOSED'  # CLOSED`
		`           , OPEN, HALF_OPEN`
		
		`    def call(self, func, *args, **`
		`       kwargs):`
		`        if self.state == 'OPEN':`
		`            if self.`
		`               _should_attempt_reset():`
		`                self.state = 'HALF_OPEN`
		`                   '`
		`            else:`
		`                raise Exception('`
		`                   Circuit␣breaker␣is␣`
		`                   OPEN')`
		
		`        try:`
		`            result = func(*args, **`
		`               kwargs)`
		`            self._on_success()`
		`            return result`
		`        except self.expected_exception`
		`           as e:`
		`            self._on_failure()`
		`            raise`
		• Apply to all inter-service calls
		• Add monitoring and alerting
		• Create fallback mechanisms
		• Document pattern usage

#	Owner	Commit Description & Detailed Tasks
27	Fouani	**Redis caching implementation**

- Add Redis to docker-compose:

```
redis:
  image: redis:7-alpine
  ports:
    - "6379:6379"
  volumes:
    - redis_data:/data
```

- Create `utils/cache.py`:

```
import redis
import json
from functools import wraps

redis_client = redis.Redis(host='redis'
   , port=6379, decode_responses=True)

def cache_result(expiration=300):
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            cache_key = f"{func.
               __name__}:{str(args)}:{
               str(kwargs)}"
            cached = redis_client.get(
               cache_key)

            if cached:
                return json.loads(
                   cached)

            result = func(*args, **
               kwargs)
            redis_client.setex(
               cache_key, expiration,
               json.dumps(result))
            return result
        return wrapper
    return decorator
```

- Apply caching to room availability checks
- Cache user session data
- Implement cache invalidation strategies

#	Owner	Commit Description & Detailed Tasks
28	Yateem	**RabbitMQ asynchronous messaging - Part 1**

- Add RabbitMQ to docker-compose:

```
rabbitmq:
  image: rabbitmq:3-management
  ports:
    - "5672:5672"
    - "15672:15672"
  environment:
    RABBITMQ_DEFAULT_USER: admin
    RABBITMQ_DEFAULT_PASS: admin
```

- Create `messaging/publisher.py`:

```
import pika
import json

class MessagePublisher:
    def __init__(self):
        self.connection = pika.
           BlockingConnection(
            pika.ConnectionParameters('
               rabbitmq'))
        self.channel = self.connection.
           channel()

    def publish_booking_created(self,
       booking_data):
        self.channel.queue_declare(
           queue='booking_notifications
           ')
        self.channel.basic_publish(
            exchange='',
            routing_key='
               booking_notifications',
            body=json.dumps(
               booking_data)
        )
```

- Implement booking notification publisher
- Create email notification queue

#	Owner	Commit Description & Detailed Tasks
29	Fouani	**Grafana dashboard setup**

- Add Grafana and Prometheus to docker-compose:

```yaml
prometheus:
  image: prom/prometheus
  ports:
    - "9090:9090"
  volumes:
    - ./prometheus.yml:/etc/prometheus/prometheus.yml

grafana:
  image: grafana/grafana
  ports:
    - "3000:3000"
  environment:
    GF_SECURITY_ADMIN_PASSWORD: admin
```

- Create `monitoring/metrics.py`:

```python
from prometheus_client import Counter, Histogram, generate_latest

booking_counter = Counter('bookings_total',
                          'Total number of bookings')
request_duration = Histogram('request_duration_seconds',
                             'Request duration')

@app.route('/metrics')
def metrics():
    return generate_latest()
```

- Configure Prometheus scraping
- Create Grafana dashboards for all services
- Add custom metrics and alerts

#	Owner	Commit Description & Detailed Tasks
30	Yateem	**RabbitMQ asynchronous messaging - Part 2** • Create `messaging/consumer.py`:

```python
import pika
import json
from email_service import send_email

class MessageConsumer:
    def __init__(self):
        self.connection = pika.BlockingConnection(
            pika.ConnectionParameters('rabbitmq'))
        self.channel = self.connection.channel()

    def process_booking_notification(self, ch, method, properties, body):
        booking = json.loads(body)
        # Send email confirmation
        send_email(
            to=booking['user_email'],
            subject='Booking Confirmation',
            body=f"Your booking for {booking['room_name']} is confirmed"
        )
        ch.basic_ack(delivery_tag=method.delivery_tag)

    def start_consuming(self):
        self.channel.queue_declare(queue='booking_notifications')
        self.channel.basic_consume(
            queue='booking_notifications',
            on_message_callback=self.process_booking_notification
        )
        self.channel.start_consuming()
```

- Create notification worker service
- Implement retry logic for failed messages
- Add dead letter queue handling

#	Owner	Commit Description & Detailed Tasks
31	Fouani	**Enhanced monitoring and alerting** • Create alert rules in Prometheus:

```
groups:
  - name: service_alerts
    rules:
      - alert: HighErrorRate
        expr: rate(http_requests_total{
           status=~"5.."}[5m]) > 0.05
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: High error rate
             detected

      - alert: ServiceDown
        expr: up == 0
        for: 1m
        labels:
          severity: critical
```

- Configure Grafana alert notifications
- Create service health dashboards
- Add business metrics tracking
- Document monitoring setup

#	Owner	Commit Description & Detailed Tasks
32	Yateem	**Security hardening and audit logging** • Implement comprehensive audit logging:

```python
from datetime import datetime
import json

class AuditLogger:
    def __init__(self, service_name):
        self.service_name =
           service_name

    def log_action(self, user_id,
       action, resource, details=None):
        audit_entry = {
            'timestamp': datetime.
               utcnow().isoformat(),
            'service': self.
               service_name,
            'user_id': user_id,
            'action': action,
            'resource': resource,
            'details': details,
            'ip_address': request.
               remote_addr
        }

        # Write to audit log file
        with open('audit.log', 'a') as
           f:
            f.write(json.dumps(
               audit_entry) + '\n')

        # Send to centralized logging
           system
        self.send_to_elk(audit_entry)
```

- Add SQL injection prevention
- Implement API rate limiting
- Add request signing between services
- Create security test suite

#	Owner	Commit Description & Detailed Tasks

33	Fouani	**Performance optimization finalization**
		• Implement database query optimization:
		`# Add database indexes`
		`class Room(db.Model):`
		`    __tablename__ = 'rooms'`
		`    __table_args__ = (`
		`        db.Index('idx_room_capacity', 'capacity'),`
		`        db.Index('idx_room_location', 'location'),`
		`        db.Index('idx_room_status', 'status'),`
		`    )`
		
		`# Optimize N+1 queries`
		`@app.route('/api/bookings')`
		`def get_bookings():`
		`    # Use eager loading`
		`    bookings = Booking.query\`
		`        .options(joinedload(Booking.room))\`
		`        .options(joinedload(Booking.user))\`
		`        .all()`
		• Add connection pooling
		• Implement query result caching
		• Add database read replicas support
		• Create performance benchmarks

2.4 Phase 5: Final Integration & Deployment (Commits 34-35)

#	Owner	Commit Description & Detailed Tasks
#	Owner	Commit Description & Detailed Tasks
34	Yateem	**Jenkins CI/CD pipeline and final integration tests** • Create Jenkinsfile:

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build') {
            steps {
                sh 'make install'
            }
        }

        stage('Test') {
            steps {
                sh 'make test'
                publishHTML(target: [
                    reportDir: '
                       coverage_html_report
                       ',
                    reportFiles: 'index
                       .html',
                    reportName: '
                       Coverage Report'
                ])
            }
        }

        stage('Build Docker Images') {
            steps {
                sh 'docker-compose
                   build'
            }
        }

        stage('Deploy') {
            steps {
                sh 'docker-compose up -
                   d'
            }
        }
    }

    post {
        always {
            sh 'docker-compose logs >
               docker_logs.txt'
            archiveArtifacts artifacts:
```

#	Owner	Commit Description & Detailed Tasks

35	Fouani	**Final documentation, report generation, and project submission prep**
		• Generate final Sphinx documentation
		• Create comprehensive README with:
		`# Smart Meeting Room Management System`
		
		`## Team Members`
		`- Yateem (Users & Bookings Services)`
		`- Fouani (Rooms & Reviews Services)`
		
		`## Quick Start`
		`1. Clone the repository`
		`` 2. Run ‘make all‘ to setup everything ``
		`3. Access services at:`
		`   - Users: http://localhost:5001`
		`   - Rooms: http://localhost:5002`
		`   - Bookings: http://localhost:5003`
		`   - Reviews: http://localhost:5004`
		`   - Grafana: http://localhost:3000`
		`   - RabbitMQ: http://localhost:15672`
		
		`## Architecture`
		`[Include architecture diagram]`
		
		`## API Documentation`
		`` Full API docs available at ‘/docs‘ endpoint ``
		
		`## Testing`
		`` Run ‘make test‘ for full test suite ``
		`` Coverage report at ‘coverage_html_report/index.html‘ ``
		• Prepare final report sections
		• Take all required screenshots
		• Create submission package
		• Final code review and cleanup

3 Detailed Implementation Guidelines

3.1 Database Schema Details

Listing 1: Complete Database Schema

```
-- Users Table
CREATE TABLE users (
```

```sql
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    role VARCHAR(20) NOT NULL CHECK (role IN ('admin', 'user', '
       facility_manager',
                                                      'moderator', '
                                                         auditor', '
                                                         service')),
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP,
    failed_login_attempts INTEGER DEFAULT 0,
    locked_until TIMESTAMP
);

-- Rooms Table
CREATE TABLE rooms (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL,
    capacity INTEGER NOT NULL CHECK (capacity > 0),
    floor INTEGER,
    building VARCHAR(50),
    location VARCHAR(200),
    equipment TEXT[], -- Array of equipment items
    amenities TEXT[], -- Array of amenities
    status VARCHAR(20) DEFAULT 'available'
           CHECK (status IN ('available', 'booked', 'maintenance'
              , 'out_of_service')),
    hourly_rate DECIMAL(10, 2),
    image_url VARCHAR(500),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Bookings Table
CREATE TABLE bookings (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    room_id INTEGER REFERENCES rooms(id) ON DELETE CASCADE,
    title VARCHAR(200) NOT NULL,
    description TEXT,
    start_time TIMESTAMP NOT NULL,
    end_time TIMESTAMP NOT NULL,
    status VARCHAR(20) DEFAULT 'confirmed'
           CHECK (status IN ('pending', 'confirmed', 'cancelled',
               'completed', 'no_show')),
    attendees INTEGER,
    is_recurring BOOLEAN DEFAULT FALSE,
    recurrence_pattern VARCHAR(20), -- daily, weekly, monthly
    recurrence_end_date DATE,
    cancellation_reason TEXT,
    cancelled_at TIMESTAMP,
    cancelled_by INTEGER REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT valid_booking_times CHECK (end_time > start_time),
    CONSTRAINT no_time_overlap EXCLUDE USING gist (
        room_id WITH =,
        tsrange(start_time, end_time) WITH &&
    ) WHERE (status != 'cancelled')
);

-- Reviews Table
CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    room_id INTEGER REFERENCES rooms(id) ON DELETE CASCADE,
    booking_id INTEGER REFERENCES bookings(id) ON DELETE CASCADE,
    rating INTEGER NOT NULL CHECK (rating >= 1 AND rating <= 5),
    title VARCHAR(200),
    comment TEXT,
    pros TEXT,
    cons TEXT,
    is_flagged BOOLEAN DEFAULT FALSE,
    flag_reason VARCHAR(200),
    flagged_by INTEGER REFERENCES users(id),
    flagged_at TIMESTAMP,
    is_hidden BOOLEAN DEFAULT FALSE,
    hidden_reason VARCHAR(200),
    helpful_count INTEGER DEFAULT 0,
    unhelpful_count INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    edited_at TIMESTAMP,
    CONSTRAINT one_review_per_booking UNIQUE(user_id, booking_id)
);

-- Audit Log Table
CREATE TABLE audit_logs (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    service VARCHAR(50) NOT NULL,
    action VARCHAR(50) NOT NULL,
    resource_type VARCHAR(50),
    resource_id INTEGER,
    old_values JSONB,
    new_values JSONB,
    ip_address INET,
    user_agent TEXT,
```

```sql
    success BOOLEAN DEFAULT TRUE,
    error_message TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create indexes for performance
CREATE INDEX idx_users_username ON users(username);
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_rooms_capacity ON rooms(capacity);
CREATE INDEX idx_rooms_status ON rooms(status);
CREATE INDEX idx_bookings_user_id ON bookings(user_id);
CREATE INDEX idx_bookings_room_id ON bookings(room_id);
CREATE INDEX idx_bookings_start_time ON bookings(start_time);
CREATE INDEX idx_bookings_status ON bookings(status);
CREATE INDEX idx_reviews_room_id ON reviews(room_id);
CREATE INDEX idx_reviews_user_id ON reviews(user_id);
CREATE INDEX idx_audit_logs_user_id ON audit_logs(user_id);
CREATE INDEX idx_audit_logs_created_at ON audit_logs(created_at);
```

3.2 API Endpoint Specifications

4 Testing Strategy

4.1 Unit Test Coverage Requirements

- Minimum 80% code coverage for all services
- Test all CRUD operations
- Test authentication and authorization
- Test input validation and sanitization
- Test error handling
- Test inter-service communication

4.2 Integration Test Scenarios

1. Complete booking workflow
2. User registration to booking to review flow
3. Admin room management workflow
4. Conflict resolution scenarios
5. Concurrent booking attempts

5 Deployment Checklist

All services running on correct ports

Database migrations completed

Environment variables configured

JWT secret keys set

Docker containers healthy

Inter-service communication verified

Postman collections complete

All tests passing (¿80% coverage)

Sphinx documentation generated

Performance profiling completed

Security measures implemented

Part II enhancements functional

GitHub repository clean

Final report prepared

6 Grading Rubric Alignment

7 Critical Success Factors

1. **Regular Commits**: Maintain consistent commit schedule
2. **Clear Documentation**: Document as you code
3. **Test First**: Write tests before implementation
4. **Security First**: Implement security from the start
5. **Performance Monitoring**: Profile early and often
6. **Code Reviews**: Review each other's code regularly
7. **Docker Testing**: Test containers after each change
8. **API Testing**: Use Postman after each endpoint

Endpoint	**Method**	**Description**	**Auth Required**
Users Service (Port 5001)			
/api/auth/register	POST	Register new user	No
/api/auth/login	POST	User login	No
/api/auth/logout	POST	User logout	Yes
/api/auth/refresh	POST	Refresh JWT token	Yes
/api/users	GET	Get all users (admin)	Yes (Admin)
/api/users/¡id¿	GET	Get user by ID	Yes
/api/users/profile	GET	Get current user profile	Yes
/api/users/profile	PUT	Update profile	Yes
/api/users/¡id¿	DELETE	Delete user (admin)	Yes (Admin)
/api/users/¡id¿/bookings	GET	Get user booking history	Yes
Rooms Service (Port 5002)			
/api/rooms	GET	List all rooms	No
/api/rooms/¡id¿	GET	Get room details	No
/api/rooms	POST	Add new room	Yes (Admin)
/api/rooms/¡id¿	PUT	Update room	Yes (Admin)
/api/rooms/¡id¿	DELETE	Delete room	Yes (Admin)
/api/rooms/available	GET	Check available rooms	No
/api/rooms/search	POST	Search rooms	No
Bookings Service (Port 5003)			
/api/bookings	GET	List bookings	Yes
/api/bookings/¡id¿	GET	Get booking details	Yes
/api/bookings	POST	Create booking	Yes
/api/bookings/¡id¿	PUT	Update booking	Yes
/api/bookings/¡id¿	DELETE	Cancel booking	Yes
/api/bookings/check	POST	Check availability	No
/api/bookings/conflicts	GET	Get conflicts	Yes (Admin)
Reviews Service (Port 5004)			
/api/reviews	POST	Submit review	Yes
/api/reviews/¡id¿	PUT	Update review	Yes
/api/reviews/¡id¿	DELETE	Delete review	Yes
/api/reviews/room/¡id¿	GET	Get room reviews	No
/api/reviews/¡id¿/flag	POST	Flag review	Yes
/api/reviews/flagged	GET	Get flagged reviews	Yes (Mod)
/api/reviews/¡id¿/moderate	PUT	Moderate review	Yes (Mod)

Criteria	**Points**	**Commits Covering This**
Project Management & Organization	10	1-5, 34-35
Service Development	40	6-16
API Documentation & Testing	10	17-20
Error Handling & Validation	5	3, 6-16
Docker Setup & Integration	5	4, 24
Performance Profiling	5	22, 33
GitHub & Version Control	5	All commits
Part II Tasks	20	26-33