Use Case Backend System - Hệ Thống Quản Lý Bãi Đỗ Xe Thông Minh

Mô Tả Tổng Quát

Backend là "não bộ" của hệ thống, xử lý tất cả logic kinh doanh: quản lý thẻ xe, xác thực người dùng (admin/user), quản lý parking slots, tính toán parking duration, lưu trữ database, giao tiếp với ESP32/UNO R4, và phục vụ API cho Frontend. Tài liệu này mô tả 10 use cases được implement. **Lưu ý:** Config Settings sẽ được implement ở version 2.0. **Tài liệu này mô tả 10 use cases được implement (không bao gồm Config Settings).**

Use Case 1: Đăng Nhập Người Dùng (Login)

Trường	Nội Dung
Tên Use Case	Đăng Nhập Người Dùng - Authentication JWT
Mô Tả	Người dùng (admin/staff) nhập username & password trên Frontend. Backend xác thực bằng bcrypt hash, cấp JWT token cho session. Người dùng dùng token để truy cập các API khác.
Tác Nhân	Người dùng (Admin hoặc Staff), Frontend, Backend API (/api/auth/login)
Mô Tả Chi Tiết	POST /api/auth/login nhận `{username, password}`. Backend query User model, so sánh password hash bằng bcrypt.check_password_hash(). Nếu match, tạo JWT token (sub=username, user_id, role, exp=24h), trả token + user info cho Frontend. Frontend lưu token ở localStorage/sessionStorage.
Điều Kiện Tiên Quyết	User account tồn tại ở database; Password được hash bằng bcrypt

Trường	Nội Dung

Luộng Chính	1. Frontend vào Login page
	2. Nhập username: "admin", password: "admin123"
	3. Frontend gọi POST /api/auth/login
	Body: `{username: "admin", password: "admin123"}`
	4. Backend nhận request
	5. Query database: User.query.filter_by(username="admin").first()
	6. Nếu user không tìm thấy → trả error 401 "Invalid credentials"
	7. Kiểm tra user.is_active (phải true)
	8. Hash password: bcrypt.check_password_hash(user.password_hash, "admin123")
	hoặc werkzeug.check_password_hash() (fallback)
	9. Nếu password mismatch → trả error 401
	10. Nếu match:
	- Tạo JWT token:
	`jwt.encode({`
	`"sub": "admin",`
	`"user_id": user.id,`
	`"username": "admin",`
	`"role": "admin",`
	`"exp": datetime.utcnow() + timedelta(hours=24)`
	`}, JWT_SECRET_KEY, algorithm="HS256")`
	11. Trả response 200:
	`{`
	`"success": true,`
	`"token": "eyJhbGc...",`
	`"user": {id, username, role, email}`
	`}`
	12. Frontend nhận, lưu token ở localStorage
	13. Frontend set Authorization header cho request tiếp theo: "Bearer {token}"

Luộng Thay Thế	A1: Oauth2 Login - Dùng Google/Facebook OAuth thay password A2: RFID Card Login - Dùng RFID card thay username/password (cho admin) A3: Biometric - Dùng face/fingerprint (nếu frontend web hỗ trợ)
Luộng Ngoài Lề	E1: User deactivated - Nếu is_active=false, reject login E2: Wrong password - Nếu hash mismatch, trả 401, không reveal "user not found" (security) E3: Expired token - Token 24h, nếu session lâu, user cần login lại
Response Format	✓ Has `success` wrapper: `{success: true, token: "...", user: {...}}`

Use Case 2: Tạo Tài Khoản Người Dùng Mới (Admin Only)

Trường	Nội Dung

Trường	Nội Dung
Tên Use Case	Tạo User Mới - Admin Create Staff Account
Mô Tả	Admin tạo tài khoản mới cho staff/user mới. Admin nhập username, password, email, full_name, role. Backend validate, hash password, lưu database, trả success.
Tác Nhân	Admin User, Frontend, Backend API (/api/users)
Mô Tả Chi Tiết	Admin vào User Management, nhấn "Create New User", nhập form, submit. Frontend gọi POST /api/users với admin JWT token. Backend kiểm tra admin_required decorator, verify input, hash password bằng bcrypt, insert User record.
Điều Kiện Tiên Quyết	Current user là admin (role='admin'); JWT token valid

Trường	Nội Dung

Luộng Chính	1. Admin vào "User Management" tab
	2. Nhấn "Create New User"
	3. Điền form:
	- username: "staff_01"
	- password: "securepass123"
	- email: "staff@parking.com"
	- full_name: "Nguyễn Văn A"
	- role: "user" (default)
	4. Nhấn "Save"
	5. Frontend gọi POST /api/users
	Header: `Authorization: Bearer {jwt_token}`
	Body: form data
	6. Backend nhận request
	7. Verify admin_required decorator → kiểm tra role='admin'
	8. Nếu không admin → trả 403 Forbidden
	9. Validate input:
	- validate_username("staff_01") → check 3-20 chars, alphanumeric+underscore
	- validate_password("securepass123") → check min 6 chars
	- Check duplicate username: User.query.filter_by(username="staff_01").first()
	- Nếu duplicate → trả 409 Conflict "Username already exists"
	10. Nếu valid:
	- Hash password: bcrypt.hashpw(password, bcrypt.gensalt())
	- Create User model:
	`new_user = User(`
	`username="staff_01",`
	`password_hash=hashed,`
	`email="staff@parking.com",`
	`full_name="Nguyễn Văn A",`
	`role="user",`
	`is_active=True`
	`)`
	- db.session.add(new_user)
	- db.session.commit()
	11. Trả success 201:
	`{`
	`"success": true,`
	`"user": {id, username, email, role}`
	`}`
	12. Frontend refresh user list, hiển thị user mới

Luộng Thay Thế	A1: Batch create - CSV upload, create nhiều user cùng lúc A2: Email invitation - Gửi email link để user tự set password A3: LDAP integration - Sync user từ LDAP/Active Directory (enterprise)

Trường	Nội Dung
Luộng Ngoài Lề	E1: Duplicate email - Nếu email đã tồn tại, reject (email should unique) E2: Invalid email format - Validate email format bằng regex E3: Weak password - Suggest strong password pattern (uppercase, number, symbol) E4: Permission denied - Nếu current user không admin, trả 403
Điều Kiện Sau	User account được tạo; Database có record mới; Admin có thể manage account này

Use Case 3: Quản Lý Danh Sách User (List, Filter, Delete)

Trường	Nội Dung
Tên Use Case	Quản Lý User - Admin View/Filter/Delete User
Mô Tả	Admin xem danh sách tất cả user, filter theo role/status, edit info, hoặc delete/deactivate user.
Tác Nhân	Admin User, Frontend, Backend API (/api/users)
Mô Tả Chi Tiết	GET /api/users lấy danh sách user với optional filter params (role, is_active). DELETE /api/users/ hoặc POST /api/users//deactivate soft-delete user.
Điều Kiện Tiên Quyết	Admin JWT token valid; Tồn tại user records ở database

Trường	Nội Dung
Luộng Chính	**GET /api/users** - List users: 1. Admin vào User Management 2. Frontend gọi GET /api/users?role=user&is_active=true 3. Backend query: `users = User.query.filter_by(role="user", is_active=True).all()` 4. Serialize users to JSON 5. Trả 200 với user list 6. Frontend display table **DELETE /api/users/{user_id}** - Hard delete: 7. Admin chọn user, nhấn "Delete" 8. Frontend confirm: "Sure?" 9. Frontend gọi DELETE /api/users/5 Header: `Authorization: Bearer {jwt_token}` 10. Backend verify admin_required 11. Query user = User.query.get(5) 12. Nếu user != null: - db.session.delete(user) - db.session.commit() - Trả 200 "User deleted" 13. Nếu user not found: - Trả 404 **POST /api/users/{user_id}/deactivate** - Soft delete: 14. Frontend gọi POST (thay vì DELETE) 15. Backend set user.is_active = False 16. db.session.commit() 17. User không thể login nhưng record vẫn lưu (audit trail)
Luộng Thay Thế	A1: Role-based filter - Filter theo admin/staff/user role A2: Search by username - Frontend search box real-time A3: Batch delete - Multi-select, delete nhiều user cùng lúc
Luộng Ngoài Lề	E1: Cannot delete self - Nếu admin delete chính mình, reject E2: Cannot delete last admin - Nếu chỉ có 1 admin, không cho delete E3: User has active card - Nếu user có thẻ active, warning hoặc block delete
Response Format	✓ Has `success` wrapper: GET → `{success: true, users: [...], count: N}`; DELETE → `{success: true, message: "User deleted"}`
Điều Kiện Sau	Admin có danh sách user; Có thể quản lý (thêm/sửa/xóa) user

Use Case 4: Quét RFID Thẻ Vào (RFID Scan API)

Trường	Nội Dung

Trường	Nội Dung
Tên Use Case	Xử Lý RFID Scan - Backend Verify Card & Update Status
Mô Tả	UNO R4 gửi RFID UID + direction (IN/OUT) đến Backend. Backend verify card tồn tại, check anti-passback, cập nhật card status (0→1 vào, 1→0 ra), tính parking_duration, log activity.
Tác Nhân	UNO R4 (Hardware), Backend API (/api/cards/scan), Database
Mô Tả Chi Tiết	POST /api/cards/scan nhận `{uid, direction}`. Backend lookup card ở database/JSON. Nếu card tồn tại: check status, direction, anti-passback rules. Nếu ok: update card status, set entry_time hoặc exit_time, tính parking_duration, log, auto-backup, trả response "entry"/"exit". Nếu không tìm: add vào unknown_cards, log warning.
Điều Kiện Tiên Quyết	Card database khả dụng (JSON hoặc SQLite); UNO R4 quét thành công

Trường	Nội Dung
Luồng Chính	1. UNO R4 quét RFID, UID="A1B2C3D4", direction="IN" 2. UNO R4 gửi POST /api/cards/scan Body: `{uid: "A1B2C3D4", direction: "IN"}` 3. Backend nhận request 4. Validate input: - validate_card_id("A1B2C3D4") → check format - direction in ["IN", "OUT"] 5. Lookup card: card = card_service.get_card("A1B2C3D4") 6. Nếu card not found: - Log "Unknown card: A1B2C3D4" - Add vào unknown_cards.json - Trả 404: `{success: false, message: "Card not found"}` - UNO R4 không mở barrier 7. Nếu card found: - Check anti-passback: - Nếu direction="IN" nhưng status=1 (already inside) → reject "Card already inside" - Nếu direction="OUT" nhưng status=0 (already outside) → reject "Card not inside" - Nếu anti-passback fail → trả 409, barrier không mở 8. Nếu anti-passback ok: - Call card.update_status(): - Nếu direction="IN": status=0→1, set entry_time=now - Nếu direction="OUT": status=1→0, set exit_time=now, calculate parking_duration - Save card_service.update_card(uid, card) - Log activity: action="entry"/"exit", card_id, direction, duration 9. Auto-backup nếu đủ interval 10. Trả 200: `{success: true, action: "entry"/"exit", duration: "2 hours 30 mins"}` 11. UNO R4 nhận, mở barrier
Luồng Thay Thế	A1: Override anti-passback - Admin force open barrier (bypass logic) A2: No status check - Không cập nhật card status, chỉ log A3: Manual card input - Staff nhập card UID thủ công thay RFID scan
Luồng Ngoài Lề	E1: Card format invalid - Validate format, reject nếu invalid E2: Duplicate scan - Nếu scan lại cùng thẻ trong 1s, bỏ qua (dung sai) E3: Clock skew - Nếu entry_time > exit_time, duration âm, fix bằng max(0, duration) E4: Unknown card alert - Email/SMS alert admin nếu unknown card scan quá 5 lần/ngày
Điều Kiện Sau	Card status cập nhật; Parking duration tính toán (nếu exit); Activity logged; Barrier mở

Use Case 5: Quản Lý Danh Sách Thẻ Xe (Card CRUD)

Trường	Nội Dung

Trường	Nội Dung
Tên Use Case	Quản Lý Thẻ Xe - Admin Add/Edit/Delete Card
Mô Tả	Admin quản lý thẻ xe: xem danh sách, tìm kiếm, thêm mới, sửa thông tin, xóa thẻ.
Tác Nhân	Admin User, Frontend, Backend API (/api/cards)
Mô Tả Chi Tiết	GET /api/cards lấy danh sách phân trang. POST /api/cards tạo card mới. PUT /api/cards/ sửa card. DELETE /api/cards/ xóa card. Tất cả yêu cầu admin JWT token.
Điều Kiện Tiên Quyết	Admin JWT token valid; Card database khả dụng

Luộng Chính	**GET /api/cards?page=1&limit=10** - List cards:
	1. Admin vào "Cards Management"
	2. Frontend gọi GET /api/cards?page=1&limit=10
	3. Backend query:
	`cards = card_service.get_all_cards()`
	Paginate: offset=(page-1)*limit, limit=10
	4. Serialize cards, tính total count
	5. Trả 200: `{success: true, cards: [...], count: 10, total: 100}`
	6. Frontend display table
	
	POST /api/cards - Create new card:
	7. Admin nhấn "Add New Card"
	8. Điền form: uid="RFID12345", name="Nguyễn Văn B", card_type="resident"
	9. Frontend gọi POST /api/cards
	Body: form data
	10. Backend validate:
	- validate_card_id(uid) → format check
	- Check duplicate: card_service.get_card(uid) != null → reject
	- name length 2-100
	- card_type in ["resident", "temporary", "guest"]
	11. Create Card model:
	`new_card = Card(uid, name, card_type, status=0, created_at=now)`
	12. Save: card_service.add_card(new_card)
	13. Auto-backup
	14. Log: action="card_created"
	15. Trả 201: `{success: true, card: {id, uid, name, card_type}}`
	16. Frontend refresh list
	
	PUT /api/cards/{card_id} - Update card:
	17. Admin chọn card, nhấn edit, thay đổi name hoặc card_type
	18. Frontend gọi PUT /api/cards/5
	Body: `{name: "Nguyễn Văn B - Updated", card_type: "temporary"}`
	19. Backend:

Trường	Nội Dung
	- Get card - Update fields - Save - Log: action="card_updated" - Trả 200: `{success: true, message: "Card updated"}` **DELETE /api/cards/{card_id}** - Delete card: 20. Admin nhấn "Delete" trên card 21. Frontend confirm, gọi DELETE /api/cards/5 22. Backend: - Get card - Move vào unknown_cards.json (audit) - Delete từ cards.json/database - Log: action="card_deleted" - Trả 200: `{success: true, message: "Card deleted"}`
Luộng Thay Thế	A1: Bulk import - CSV upload, import 100 card cùng lúc A2: Card expiry - Thẻ có thời hạn (expire date), auto-disable A3: Card suspension - Suspend card (blacklist) thay delete
Luộng Ngoài Lề	E1: Cannot delete active card - Nếu card đang inside (status=1), ask admin confirm E2: Duplicate UID - Nếu UID duplicate, reject E3: Invalid card_type - Reject nếu card_type không hợp lệ
Response Format	✓ Has `success` wrapper: GET → `{success: true, cards: [...], count: N, total: N}`; POST → `{success: true, card: {...}}`; PUT/DELETE → `{success: true, message: "..."}`
Điều Kiện Sau	Card được thêm/sửa/xóa thành công; Database cập nhật; Frontend refresh

Use Case 6: Lấy Dữ Liệu Cảm Biến Parking Slots

Trường	Nội Dung
Tên Use Case	Lấy Trạng Thái Parking Slots - Pull từ ESP32
Mô Tả	Frontend định kỳ gọi GET /api/parking-slots/ để lấy dữ liệu 6 cảm biến từ ESP32. Backend fetch từ ESP32 HTTP endpoint, parse, validate, cache, trả về Frontend. Frontend hiển thị grid 6 slots.
Tác Nhân	Frontend, Backend API (/api/parking-slots), ESP32

Trường	Nội Dung
Mô Tả Chi Tiết	GET /api/parking-slots gọi ESP32Service.get_parking_slots() → HTTP GET http://192.168.4.5:80/data. ESP32 trả JSON `{data: [0,1,0,1,0,0], ...}`. Backend parse vào ParkingSlotsData model, validate, tính summary (occupied, available, occupancy_rate). Cache result 5 phút. Trả Frontend.
Điều Kiện Tiên Quyết	ESP32 đang chạy WebServer ở port 80; Cảm biến đã quét; Frontend User/Admin

Luộng Chính	1. Frontend mở "Parking Dashboard"
	2. Frontend gọi GET /api/parking-slots/ (mỗi 10-30s polling)
	3. Backend check cache:
	- Nếu cache valid (< 5 phút) → return cached data
	- Nếu cache expired → fetch mới
	4. Gọi ESP32Service.get_parking_slots()
	5. HTTP GET http://192.168.4.5:80/data với timeout 10s
	6. Nếu fail (timeout, connection error) → trả 503/504, return cached data
	7. Nếu success, parse JSON:
	`{`
	`"success": true,`
	`"data": [0, 1, 0, 1, 0, 0], // 6 slots`
	`"soIC": 1,`
	`"totalSensors": 6,`
	`"timestamp": 1703596800`
	`}`
	8. Create ParkingSlotsData object
	9. Validate: tất cả 6 slots present, status in [0,1]
	10. Tính summary:
	- occupied = count(status=1) = 3
	- available = 6 - occupied = 3
	- occupancy_rate = 3/6 * 100 = 50%
	11. Cache result
	12. Trả 200: `{success: true, data: {...}, summary: {...}}`
	13. Frontend parse, update UI:
	- Grid 6 slots, green=empty, red=occupied
	- Summary: "3/6 slots occupied, 50%"
	- Last update timestamp

Luộng Thay Thế	A1: Push model - ESP32 push data mỗi 30s, Frontend subscribe WebSocket A2: Direct ESP32 - Frontend direct GET ESP32 (CORS may block) A3: Database cache - Query database time-series history thay call ESP32

Trường	Nội Dung
Luộng Ngoài Lề	E1: ESP32 offline - HTTP timeout 10s, return cached/empty, status 503 E2: Data validation fail - Nếu data không valid, log error, return previous valid data E3: Cache stale - Nếu cache > 5 min, try refresh ESP32, fallback cache E4: Network slow - HTTP call slow, may impact response time, client có timeout
Điều Kiện Sau	Frontend có dữ liệu parking slots mới nhất; Grid 6 slots cập nhật

Use Case 7: Reset Cảm Biến ESP32 (Admin Remote Control)

Trường	Nội Dung
Tên Use Case	Reset Cảm Biến - Admin Command Reset từ Frontend
Mô Tả	Admin nhấn nút "Reset Sensors" trên Frontend. Frontend gọi POST /api/parking-slots/reset. Backend forward POST /reset đến ESP32. ESP32 hiệu chỉnh cảm biến. Backend trả kết quả cho Frontend.
Tác Nhân	Admin User, Frontend, Backend API (/api/parking-slots/reset), ESP32
Mô Tả Chi Tiết	Require admin role. HTTP POST command forward ESP32 reset endpoint. ESP32 clear sensor baseline, reinit. Return status.
Điều Kiện Tiên Quyết	Admin JWT token valid; ESP32 đang chạy
Luộng Chính	1. Admin vào Dashboard, thấy cảm biến không chính xác 2. Admin nhấn "Reset Sensors" 3. Frontend gọi POST /api/parking-slots/reset Header: `Authorization: Bearer {jwt_token}` 4. Backend verify admin_required → nếu không admin trả 403 5. Call ESP32Service.reset_sensors() 6. HTTP POST http://192.168.4.5:80/reset với timeout 10s 7. Nếu fail → trả 503 "ESP32 offline" 8. Nếu success, ESP32 trả: `{success: true, message: "Sensors reset successfully"}` 9. Backend parse, trả 200: `{success: true, message: "..."}` 10. Frontend hiển thị notification "Reset Successful" 11. Auto-refresh parking slots data
Luộng Thay Thế	A1: Automatic reset - System auto reset nếu detect anomaly A2: Threshold adjustment - Admin adjust detection threshold thay full reset

Trường	Nội Dung
Luộng Ngoài Lề	E1: User không admin - Trả 403 Forbidden E2: ESP32 fail reset - ESP32 trả success:false, Backend log warning
Điều Kiện Sau	Cảm biến reset, baseline clear; Accuracy cải thiện

Use Case 8: Xem Lịch Sử Hoạt Động (Activity Logs)

Trường	Nội Dung
Tên Use Case	Xem Activity Logs - Admin/User View RFID/Card Events
Mô Tả	Người dùng xem lịch sử hoạt động: thẻ vào/ra, thời gian, parking duration, unknown cards. Filter theo ngày, card, action type.
Tác Nhân	Admin/User, Frontend, Backend API (/api/cards/logs)
Mô Tả Chi Tiết	GET /api/cards/logs?card_id=xxx&action=entry&date_from=...&date_to=... query CardLog từ database (hoặc JSON). Return paginated list. Frontend display table với timestamp, card, action, duration, details.
Điều Kiện Tiên Quyết	Activity logs lưu ở database hoặc JSON; User JWT token valid

Luộng Chính	1. User vào "Activity Logs" tab
	2. Filter: date_from="2025-01-01", date_to="2025-01-31", card_id="A1B2C3D4"
	3. Frontend gọi GET /api/cards/logs?...filters...
	4. Backend query CardLog:
	`logs = CardLog.query.filter(`
	`CardLog.timestamp >= date_from,`
	`CardLog.timestamp <= date_to,`
	`CardLog.card_id == "A1B2C3D4"`
	`).order_by(CardLog.timestamp.desc()).limit(100).all()`
	5. Serialize logs to JSON
	6. Trả 200: `{success: true, logs: [...], total: 50, count: 50}`
	7. Frontend display table:
	- Columns: timestamp, card_id, action (entry/exit), parking_duration, details
	- Example row: "2025-01-15 08:30:00

Luộng Thay Thế	A1: Real-time log stream - WebSocket push logs khi xảy ra A2: Aggregated report - Daily/weekly/monthly summary report A3: Analytics chart - Biểu đồ busy hour, occupancy trend

Trường	Nội Dung
Luộng Ngoài Lề	E1: No logs - Nếu không có log match filter, return empty list E2: Large dataset - Nếu log quá nhiều (> 10k), paginate để tránh slow query E3: Permission - User chỉ thấy log của chính mình, admin thấy tất cả (phụ thuộc policy)
Response Format	✓ Has `success` wrapper: `{success: true, logs: [...], total: N, count: N}`
Điều Kiện Sau	User xem được lịch sử hoạt động; Có thể audit, report

Use Case 9: Thống Kê Hệ Thống (Statistics & Analytics)

Trường	Nội Dung
Tên Use Case	Thống Kê Hệ Thống - Dashboard Stats
Mô Tả	Dashboard hiển thị thống kê: tổng thẻ, số thẻ inside, số thẻ outside, occupancy rate, unknown cards, average parking duration, busy hours. Data từ database aggregation.
Tác Nhân	Admin/User, Frontend, Backend API (/api/cards/statistics)
Mô Tả Chi Tiết	GET /api/cards/statistics query Card model, aggregate: count, occupied, occupancy_rate, etc. Cache result 5 phút. Trả Frontend. Frontend render cards/charts.
Điều Kiện Tiên Quyết	Card database có data; User JWT token valid
Luộng Chính	1. User vào "Dashboard" 2. Frontend gọi GET /api/cards/statistics 3. Backend query: - total_cards = Card.query.count() - inside_cards = Card.query.filter_by(status=1).count() - outside_cards = total_cards - inside_cards - occupancy_rate = inside_cards / total_cards * 100 - unknown_count = UnknownCard.query.count() - avg_duration = avg(parking_duration) từ CardLog 4. Tính top busy hours: group CardLog by hour, count entry/exit 5. Cache result 5 phút 6. Trả 200: `{success: true, data: {total_cards, inside_cards, outside_cards, occupancy_rate, unknown_count, avg_duration, busy_hours: [...]}}` 7. Frontend display: - KPI cards: "Total Cards: 100", "Inside: 45 (45%)", "Outside: 55 (55%)" - Chart occupancy_rate vs time (24h) - Chart busy hours (bar chart) - Alerts: "Unknown cards: 5", "Avg duration: 2h 30m"

Trường	Nội Dung
Luộng Thay Thế	A1: Real-time stats - WebSocket push updates A2: Custom date range - User select date range, regenerate stats A3: Export report - PDF/Excel export của stats
Luộng Ngoài Lề	E1: No data - Nếu database empty, return 0s E2: Query slow - Nếu table lớn, aggregation chậm, use pre-computed materialized view E3: Timezone issue - Timestamp UTC, convert client-side để correct display
Response Format	✓ Has `success` wrapper: `{success: true, data: {total_cards, inside_cards, outside_cards, occupancy_rate, unknown_count, avg_duration, busy_hours}}`
Điều Kiện Sau	Dashboard hiển thị statistics; User có insight về parking usage

Use Case 10: Auto Backup Database (Scheduled Task)

Trường	Nội Dung
Tên Use Case	Auto Backup - Hệ Thống Tự Động Backup Dữ Liệu
Mô Tả	Backend scheduler chạy mỗi 1 giờ, tự động backup cards.json, unknown_cards.json, database. Lưu vào backups/ folder. Xóa backup cũ hơn 24h.
Tác Nhân	Backend Scheduler (ScheduledTasks), BackupService
Mô Tả Chi Tiết	ScheduledTasks.start_scheduler() launch daemon thread. Mỗi 3600s, gọi create_hourly_backup(). BackupService.create_backup(file_path) copy file sang backups/ với timestamp. BackupService.cleanup_old_backups(max_count=5) xóa file cũ.
Điều Kiện Tiên Quyết	Backend running; Disk space available; Scheduler initialized

Trường	Nội Dung
Luộng Chính	1. Backend start_scheduler() 2. Daemon thread launched, loop 3. Mỗi 3600s (1 giờ): - Call create_hourly_backup() 4. BackupService.create_backup(CARDS_FILE): - Source: backend/data/cards.json - Destination: backend/data/backups/cards.json.backup_20250115_143000 - Copy file - Verify file copied successfully 5. Repeat cho unknown_cards.json 6. Database backup (optional): dump SQLite database - `sqlite3 parking_system.db ".dump" > backups/parking_system.db.backup_20250115_143000.sql` 7. cleanup_old_backups(max_count=5): - List tất cả backup files - Sort by modification time - Xóa những file cũ hơn 5 backups mới nhất 8. Log: "Backup created at 2025-01-15 14:30:00" 9. Sleep 3600s, loop lại
Luộng Thay Thế	A1: Manual backup - Admin click "Backup Now" button A2: Cloud backup - Backup sang AWS S3 / Google Drive A3: Real-time replication - Sync to replica database thay snapshot backup
Luộng Ngoài Lề	E1: Backup fail - Disk full, permission denied, log error nhưng không crash E2: Old backup delete fail - Nếu cleanup fail, log warning, backup vẫn lưu E3: Backup too large - Nếu database lớn, compress vào .gz
Điều Kiện Sau	Backup file tạo mới ở backups/ folder; Dữ liệu được bảo vệ

Bảng Tóm Tắt User Key (By Role)

Admin User Keys:

User Key	Use Case	Hành Động
Login/Auth	UC-1	Nhập username/password, nhận JWT token
User Management	UC-2, UC-3	Create/edit/delete user account
Card Management	UC-5	Add/edit/delete card, bulk import
RFID Processing	UC-4	Verify card, update status, log activity
Parking Monitoring	UC-6, UC-7	View slots, reset sensors
Activity Logs	UC-8	View entry/exit logs, filter, export

User Key	Use Case	Hành Động
Dashboard Stats	UC-9	View system statistics, analytics
Backup Management	UC-10	Manual backup, restore, cleanup

Regular User Keys:

User Key	Use Case	Hành Động
Login/Auth	UC-1	Nhập username/password, nhận JWT token
View Dashboard	UC-6, UC-8, UC-9	View parking status, logs, stats (read-only)
Cannot:	UC-2, UC-3, UC-5, UC-7	Không thể tạo/edit/delete user hoặc card

Kết Luận

Backend là trung tâm xử lý logic, quản lý user (admin/user), thẻ xe, RFID events, parking slots data, logging, backup. Sử dụng JWT auth để protect endpoints. Giao tiếp với ESP32/UNO R4 qua HTTP. Phục vụ API cho Frontend. Lưu trữ data ở SQLite database + JSON legacy files.