Phase 5: Lead → Resident Conversion Workflow

Date: December 9, 2025
Project: CareLinkAI
Status: 🔵 **PLANNING PHASE**
Dependencies: Phase 4 RBAC (✅ Deployed)

📋 Overview

Purpose

Create a smooth, structured workflow for converting leads (inquiries) into residents, with proper status tracking, data transfer, and permission management.

Business Value

- **Streamlined Onboarding**: Reduce manual data entry and errors
- **Status Tracking**: Clear visibility into conversion pipeline
- **Compliance Ready**: Ensure all required information collected
- **RBAC Integration**: Leverage Phase 4 permissions for secure conversions

Success Criteria

- One-click conversion from lead to resident
- Automatic data transfer (contact info, family details, care requirements)
- Status workflow with approval stages
- Audit trail for all conversions
- Permission-based conversion actions

🎯 Key Features

1. Lead Conversion Workflow

Current State (Inquiry/Lead System)

- Leads exist in “Inquiries” or “Leads” section

- Manual process to create resident from lead
- No structured workflow for conversion
- Data duplication between lead and resident

Target State (Phase 5)

- **Convert to Resident** button on lead detail page
- Pre-population of resident form with lead data
- Status transitions: `New Lead → Qualified → Converting → Resident`
- Automatic family contact creation from lead contact
- Audit log for conversion actions

2. Conversion Statuses

Lead Statuses

- `NEW` - Initial inquiry received
- `CONTACTED` - Operator reached out
- `QUALIFIED` - Meets admission criteria
- `CONVERTING` - Conversion in progress
- `CONVERTED` - Resident created
- `DECLINED` - Lead declined service
- `LOST` - Lead went elsewhere

Conversion Workflow

```
NEW → CONTACTED → QUALIFIED → [Convert] → CONVERTING → RESIDENT CREATED
                                    ↓
                                DECLINED/LOST
```

3. Data Mapping

Lead → Resident Mapping

Lead Field	Resident Field	Notes
`name`	`firstName` + `lastName`	Split full name
`email`	N/A	Transfer to family contact
`phone`	N/A	Transfer to family contact
`careNeeds`	`careLevel`	Map care description to level
`preferredMoveInDate`	`admissionDate`	Use as target admission
`budget`	N/A	Store in notes or custom field
`homePreference`	`homeId`	Assign to preferred home

Family Contact Creation

- Automatically create family contact from lead contact info
- Set as primary contact
- Transfer relationship info
- Set permission level based on lead preferences

4. Permission-Based Conversion

Who Can Convert Leads?

- **Admin**: Convert any lead to any home
- **Operator**: Convert leads to their managed homes
- **Caregiver**: No conversion access
- **Family**: No conversion access

New Permission

```
PERMISSIONS.LEADS_CONVERT = 'leads.convert';

ROLE_PERMISSIONS[UserRole.ADMIN] = [...existing,
        PERMISSIONS.LEADS_CONVERT];
ROLE_PERMISSIONS[UserRole.OPERATOR] = [...existing,
        PERMISSIONS.LEADS_CONVERT];
```

🏗 Technical Architecture

Database Changes

1. Add Conversion Tracking to Lead Model

```
model Lead {
  id                String           @id @default(cuid())
  // ... existing fields ...

  // New fields for conversion
  conversionStatus  ConversionStatus @default(NEW)
  qualifiedAt       DateTime?
  convertingAt      DateTime?
  convertedAt       DateTime?
  convertedById     String?
  convertedBy       User?            @relation("LeadConverter",
fields: [convertedById], references: [id])
  residentId        String?          @unique
  resident          Resident?        @relation(fields: [residentId],
references: [id])
```

```
  // Conversion metadata
  conversionNotes   String?          @db.Text
  declineReason     String?

  @@index([conversionStatus])
  @@index([convertedAt])
}

enum ConversionStatus {
  NEW
  CONTACTED
  QUALIFIED
  CONVERTING
  CONVERTED
  DECLINED
  LOST
}
```

2. Add Conversion Reference to Resident Model

```
model Resident {
  id                String           @id @default(cuid())
  // ... existing fields ...

  // New field for conversion tracking
  sourceLeadId      String?          @unique
  sourceLead        Lead?            @relation("LeadConversion")

  @@index([sourceLeadId])
}
```

API Endpoints

1. Convert Lead to Resident

```
POST /api/leads/[id]/convert

Request Body:
{
  homeId: string,
  admissionDate: Date,
  careLevel: string,
  roomNumber?: string,
  specialInstructions?: string
}

Response:
{
```

```
  success: true,
  residentId: string,
  familyContactId: string,
  message: "Lead converted to resident successfully"
}

Permissions: PERMISSIONS.LEADS_CONVERT
Data Scoping: Can only convert leads for homes they manage
```

2. Update Conversion Status

```
PATCH /api/leads/[id]/status

Request Body:
{
  status: ConversionStatus,
  notes?: string,
  declineReason?: string
}

Response:
{
  success: true,
  lead: Lead
}

Permissions: PERMISSIONS.LEADS_UPDATE
```

3. Get Conversion Pipeline

```
GET /api/leads/pipeline

Query Parameters:
- homeId?: string
- status?: ConversionStatus[]

Response:
{
  pipeline: {
    new: number,
    contacted: number,
    qualified: number,
    converting: number,
    converted: number,
    conversionRate: number
  },
  leads: Lead[]
}
```

```
Permissions: PERMISSIONS.LEADS_VIEW
Data Scoping: Returns pipeline for user's accessible homes
```

UI Components

1. Lead Detail Page Updates

File: `src/app/operator/leads/[id]/page.tsx`

New Features: - Status badge with conversion workflow - **“Convert to Resident”** button (permission-gated) - Conversion progress indicator - Quick view of what data will transfer - Link to created resident after conversion

2. Conversion Modal

Component: `ConvertLeadModal.tsx`

Features: - Pre-populated form with lead data - Home selection (scoped to user’s homes) - Admission date picker - Care level selection - Room assignment - Family contact configuration - Review & confirm step

3. Conversion Pipeline Dashboard

Component: `ConversionPipelineDashboard.tsx`

Features: - Kanban-style board with status columns - Drag-and-drop status updates (if time permits) - Conversion metrics (rate, time-to-convert) - Filter by home, date range - Quick actions on each lead card

4. Resident Profile Enhancement

File: `src/app/operator/residents/[id]/page.tsx`

New Section: - “Created from Lead” badge (if converted) - Link back to original lead - Conversion date and by whom - Original inquiry details

---

🔄 Conversion Workflow Logic

Step-by-Step Conversion Process

1. Validation

```
async function validateLeadConversion(leadId: string, userId:
        string) {
```

```
  // Check lead exists and is qualified
  const lead = await prisma.lead.findUnique({ where: { id: leadId }
        });
  if (!lead) throw new Error('Lead not found');
  if (lead.conversionStatus === 'CONVERTED') {
    throw new Error('Lead already converted');
  }
  if (lead.conversionStatus !== 'QUALIFIED') {
    throw new Error('Lead must be qualified before conversion');
  }

  // Check user has permission
  await requirePermission(userId, PERMISSIONS.LEADS_CONVERT);

  // Check user has access to target home
  await requireHomeAccess(userId, targetHomeId);
}
```

2. Data Transfer

```
async function convertLeadToResident(leadId: string, conversionData:
        ConversionData) {
  return await prisma.$transaction(async (tx) => {
    // 1. Create resident
    const resident = await tx.resident.create({
      data: {
        firstName: parseFirstName(lead.name),
        lastName: parseLastName(lead.name),
        homeId: conversionData.homeId,
        admissionDate: conversionData.admissionDate,
        careLevel: conversionData.careLevel,
        roomNumber: conversionData.roomNumber,
        status: 'ACTIVE',
        sourceLeadId: leadId,
        // ... other fields
      }
    });

    // 2. Create family contact
    const familyContact = await tx.familyContact.create({
      data: {
        residentId: resident.id,
        name: lead.contactName || lead.name,
        relationship: lead.relationship || 'Family Member',
        phone: lead.phone,
        email: lead.email,
        isPrimary: true,
        permissionLevel: 'FULL_ACCESS',
        contactPreference: 'ANY',
```

```
      }
    });

    // 3. Update lead status
    await tx.lead.update({
      where: { id: leadId },
      data: {
        conversionStatus: 'CONVERTED',
        convertedAt: new Date(),
        convertedById: userId,
        residentId: resident.id,
        conversionNotes: conversionData.notes,
      }
    });

    // 4. Create audit log
    await createAuditLog({
      action: AuditAction.LEAD_CONVERTED,
      userId,
      leadId,
      residentId: resident.id,
      details: {
        leadName: lead.name,
        residentName: `${resident.firstName} ${resident.lastName}`,
        homeId: conversionData.homeId,
      }
    });

    return { resident, familyContact };
  });
}
```

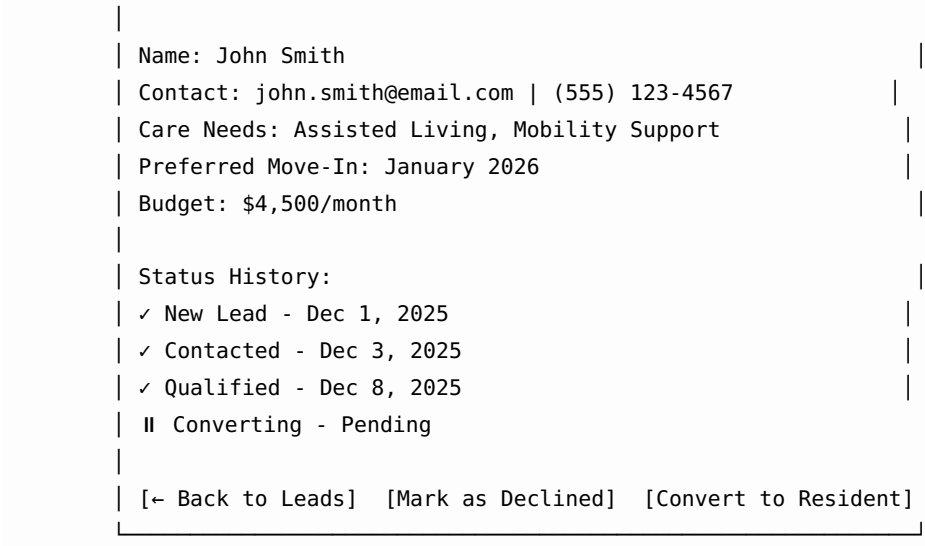
3. Post-Conversion Actions

- Send notification to operator/admin
- Send welcome email to family
- Create initial care plan (future phase)
- Schedule first assessment (future phase)

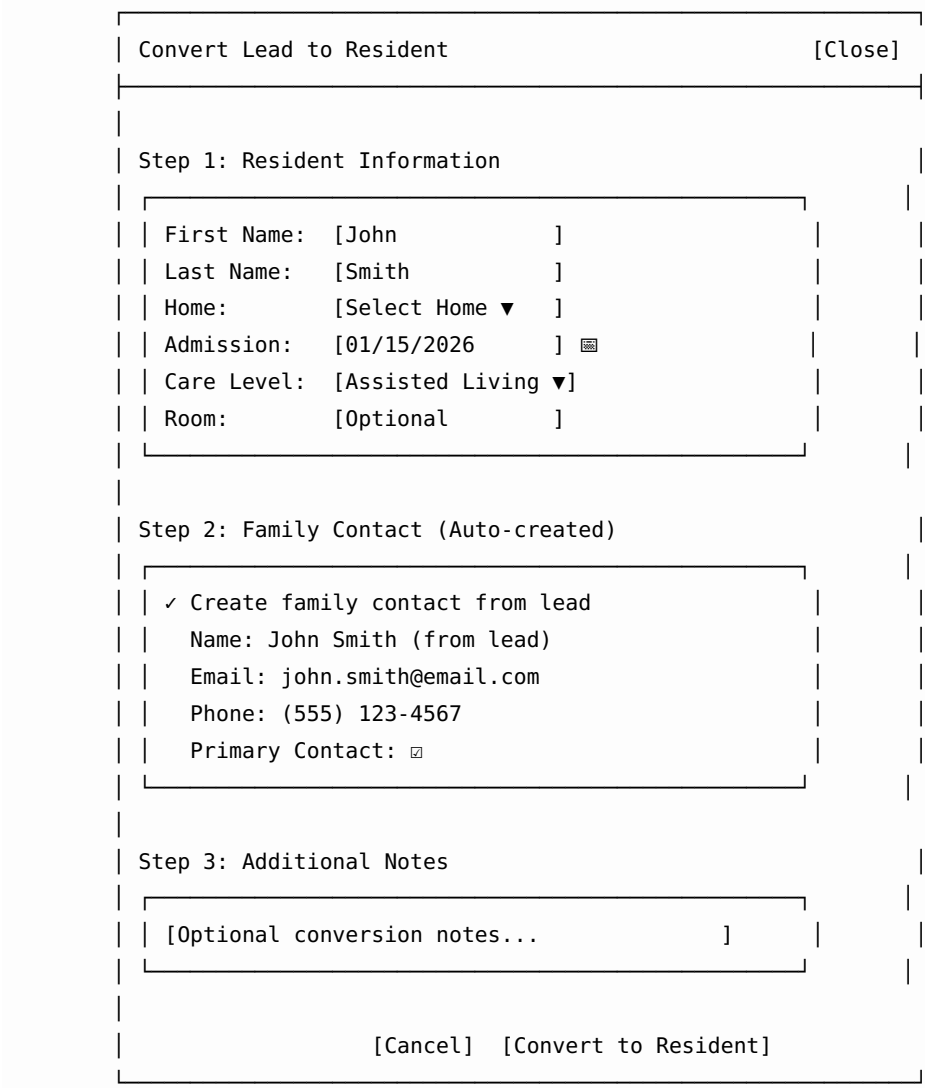
📊 UI Mockups & User Experience

1. Lead Detail Page (Before Conversion)

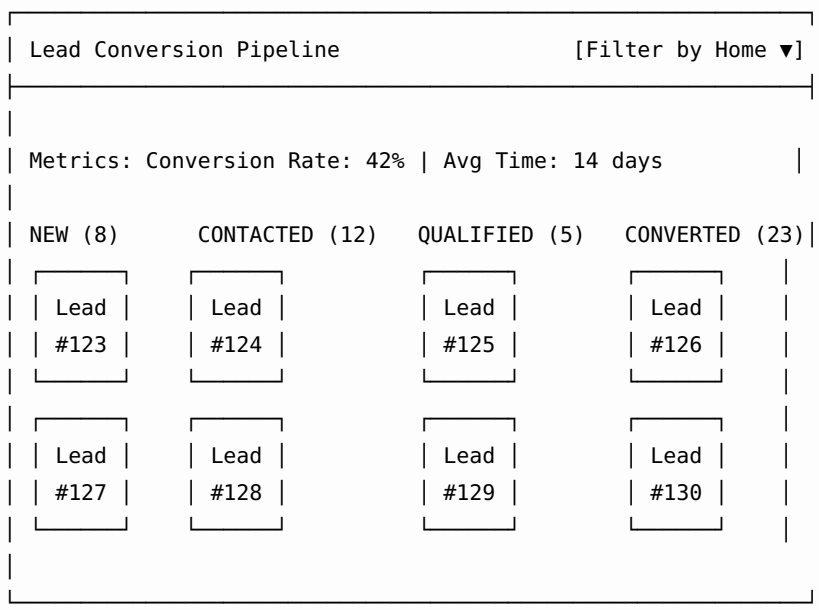
```
┌──────────────────────────────────────────────────────────────┐
│ Lead #12345                          [Qualified] [Convert]   │
├──────────────────────────────────────────────────────────────┤
```



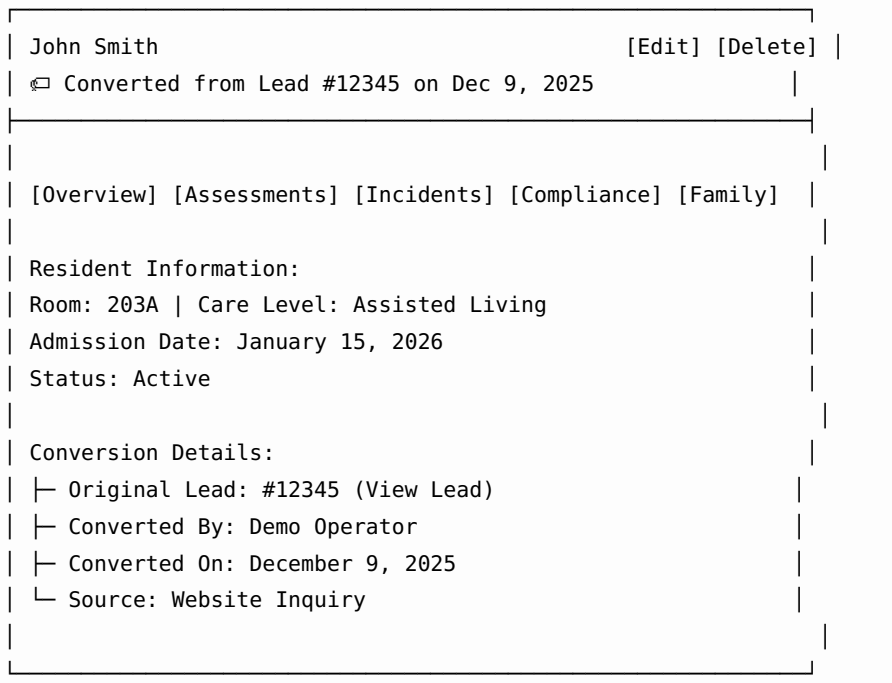
2. Conversion Modal



3. Conversion Pipeline Dashboard



4. Resident Profile (After Conversion)



🧪 Testing Strategy

Unit Tests

- Conversion validation logic
- Data mapping functions
- Permission checks
- Transaction rollback on errors

Integration Tests

- API endpoint `/api/leads/[id]/convert`
- Database transaction integrity
- Audit log creation
- Family contact creation

E2E Tests (Playwright)

```
test('Admin can convert qualified lead to resident', async ({ page
        }) => {
  // Login as admin
  await loginAsAdmin(page);

  // Navigate to leads
  await page.goto('/operator/leads');

  // Click on qualified lead
  await page.click('[data-testid="lead-12345"]');

  // Verify "Convert to Resident" button visible
  await expect(page.locator('[data-testid="convert-
        button"]')).toBeVisible();

  // Click convert
  await page.click('[data-testid="convert-button"]');

  // Fill conversion form
  await page.selectOption('[name="homeId"]', 'home-1');
  await page.fill('[name="admissionDate"]', '2026-01-15');
  await page.selectOption('[name="careLevel"]', 'ASSISTED_LIVING');

  // Submit
  await page.click('[data-testid="submit-conversion"]');

  // Verify success
  await expect(page.locator('text=Lead converted
        successfully')).toBeVisible();

  // Verify redirect to resident detail
  await expect(page.url()).toContain('/operator/residents/');

  // Verify "Converted from Lead" badge
```

```
  await expect(page.locator('text=Converted from
        Lead')).toBeVisible();
});

test('Caregiver cannot see convert button', async ({ page }) => {
  await loginAsCaregiver(page);
  await page.goto('/operator/leads/12345');
  await expect(page.locator('[data-testid="convert-
        button"]')).not.toBeVisible();
});

test('Operator can only convert to their homes', async ({ page }) =>
        {
  await loginAsOperator(page);
  await page.goto('/operator/leads/12345');
  await page.click('[data-testid="convert-button"]');

  // Verify only operator's homes in dropdown
  const homeOptions = await page.locator('[name="homeId"]
        option').allTextContents();
  expect(homeOptions).toContain('Operator Home 1');
  expect(homeOptions).not.toContain('Other Operator Home');
});
```

🗓 Implementation Timeline

Week 1: Database & Backend (Days 1-3)

- ☒ **Day 1**: Database schema updates
  - Add `conversionStatus` to Lead model
  - Add `sourceLeadId` to Resident model
  - Create migration
- ☒ **Day 2**: API endpoints
  - `POST /api/leads/[id]/convert`
  - `PATCH /api/leads/[id]/status`
  - `GET /api/leads/pipeline`
- ☒ **Day 3**: Conversion logic
  - Data mapping functions
  - Transaction handling
  - Audit logging

Week 1: Frontend (Days 4-5)

- ☐ **Day 4**: Lead detail updates
  - Add status workflow UI
  - Add “Convert to Resident” button
  - Permission-based visibility
- ☐ **Day 5**: Conversion modal

- Build ConvertLeadModal component
- Form validation
- Pre-population logic

Week 2: Pipeline & Testing (Days 1-3)

☐ **Day 1**: Pipeline dashboard
- Build ConversionPipelineDashboard
- Metrics display
- Status columns

☐ **Day 2**: Testing
- Unit tests for conversion logic
- E2E tests for conversion flow
- Permission validation tests

☐ **Day 3**: Documentation & deployment
- User guide
- API documentation
- Deploy to production

🚨 Risks & Mitigation

Risk 1: Data Loss During Conversion

Impact: High
Probability: Medium
Mitigation: - Use database transactions - Rollback on any error - Log all conversion attempts - Keep lead data after conversion (don't delete)

Risk 2: Permission Bypasses

Impact: High
Probability: Low
Mitigation: - Enforce permissions at API level - Validate home access on conversion - Audit all conversion actions - Regular security testing

Risk 3: Name Parsing Errors

Impact: Medium
Probability: High
Mitigation: - Manual review in conversion modal - Allow user to edit parsed names - Handle edge cases (single name, multiple middle names) - Fallback to full name if parsing fails

Risk 4: Duplicate Residents

Impact: Medium
Probability: Medium
Mitigation: - Check for existing resident with same name + home - Show warning if potential duplicate found - Allow operator to link to existing resident instead - Unique constraint on `sourceLeadId`

🔗 Integration Points

With Phase 4 (RBAC)

- **Permissions**: New `PERMISSIONS.LEADS_CONVERT`
- **Data Scoping**: `getUserScope()` for home access
- **UI Guards**: `useHasPermission()` for button visibility
- **API Protection**: `requirePermission()` and `requireHomeAccess()`

With Existing Systems

- **Leads/Inquiries**: Read lead data for conversion
- **Residents**: Create new resident records
- **Family Contacts**: Auto-create from lead contact
- **Audit Logs**: Track conversion actions
- **Notifications**: Alert operators of conversions (future)

Future Phases

- **Phase 6**: Care plan creation on conversion
- **Phase 7**: Initial assessment scheduling
- **Phase 8**: Automated welcome communications
- **Phase 9**: Financial setup and billing integration

🖨 Documentation Deliverables

Technical Docs

- [x] This planning document
- [ ] API endpoint specifications
- [ ] Database schema documentation
- [ ] Conversion workflow diagram

User Docs

- [ ] Conversion user guide (operators)
- [ ] Pipeline dashboard guide
- [ ] FAQ for common conversion issues

- [ ] Training materials

Testing Docs

- [ ] Test plan
- [ ] Test cases
- [ ] Test data setup guide
- [ ] Validation checklist

✔ Success Metrics

Functional Metrics

- ✓ Lead converts to resident with one click
- ✓ All lead data transfers correctly
- ✓ Family contact auto-created
- ✓ Conversion tracked in audit log
- ✓ Permission-based conversion access

Performance Metrics

- Conversion time: < 2 seconds
- No data loss: 100%
- Transaction success rate: > 99%
- UI responsiveness: < 500ms

Business Metrics

- Reduced onboarding time: Target 50%
- Reduced data entry errors: Target 75%
- Improved conversion tracking: 100% visibility
- User satisfaction: Target 4.5/5 stars

🎯 Definition of Done

Backend

- [x] Database migrations applied
- [ ] All API endpoints implemented and tested
- [ ] Conversion logic with transaction handling
- [ ] Audit logging for all conversions
- [ ] Permission enforcement at API level
- [ ] Unit tests passing (90%+ coverage)

Frontend

- [ ] Lead detail page updated with conversion button
- [ ] Conversion modal fully functional
- [ ] Pipeline dashboard implemented
- [ ] Resident profile shows conversion origin
- [ ] Permission-based UI visibility
- [ ] Responsive design (mobile + desktop)

Testing

- [ ] Unit tests written and passing
- [ ] Integration tests for API endpoints
- [ ] E2E tests for conversion flow
- [ ] Permission validation tests
- [ ] Manual QA completed

Documentation

- [ ] Technical documentation complete
- [ ] User guides published
- [ ] API docs updated
- [ ] Code comments added

Deployment

- [ ] Production deployment successful
- [ ] Manual validation completed
- [ ] Performance metrics verified
- [ ] No critical bugs reported

🚀 Ready to Implement

Phase 4 Status: ✅ Deployed to Production
Phase 5 Status: 🟡 Planning Complete, Ready to Start

Next Step: Begin Phase 5 implementation starting with database schema updates and API endpoints.

Estimated Duration: 2 weeks (10 working days)
Confidence Level: 90% (well-defined requirements, clear dependencies)

End of Phase 5 Planning Document