BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
WORK INTEGRATED LEARNING PROGRAMMES
COURSE HANDSON LAB ASSIGNMENT

M.Tech

Course Title	**Database Systems and Applications**
Course No(s)	**SESAP ZC337**
Lab Session	**Nov-Dec 25**

Faculty: Balachandra A, Guest Faculty, BITS Pilani (WILP) Division
Email: balachandra.ananatharamaiah@wilp.bits-pilani.ac.in
Mob: 9113656626 / 9480475967

SOFTWARE REQUIREMENTS SPECIFICATION (SRS) DOCUMENT

For EMPLOYEE PROFILE PHOTO EXTENSION - MULTIMODAL MODULE

Version: 1.0
Prepared by: Divyansh, Student ID: 2024SL70022
Date: November 19, 2025
Extension Type: Image/Document Storage (BLOB Implementation)

Contents

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) defines the requirements for the **Employee Profile Photo Extension**, a multimodal module designed to extend the existing Employee Management Database System with image storage capabilities. The extension adds Employee Profile Photo functionality using BLOB data types, demonstrating advanced database features and fulfilling the multimodal provisions designed in Term I. This document specifies all functional and non-functional requirements for the image extension, database schema modifications, and integration procedures. It serves as a reference for database developers implementing multimodal data capabilities and system evaluators assessing the extension's compliance with the base system architecture.

1.2 Scope

The Employee Profile Photo Extension automates image management for employee identification within the company database system.

Functions include:

- Upload and storage of employee profile photographs
- Secure image retrieval and display operations
- File validation and size restriction enforcement
- Integration with existing Employee Management CRUD operations

Extension is a multimodal database enhancement supporting:

- BLOB data type implementation within existing BCNF schema
- Backward compatibility with all existing system operations
- Standard image format support (JPEG, PNG) with security validation
- **The extension maintains data integrity, referential consistency, and BCNF normalization**
- **Demonstrates practical implementation of multimodal database provisions from Term I design**

1.3 Extension Rationale

This implementation demonstrates the **future-ready multimodal architecture** designed in Term I, specifically utilizing the BLOB storage provisions for Image/Document data as outlined in the original system design. The extension validates the extensibility principles built into the normalized schema and provides a foundation for additional multimodal capabilities.

1.4 References

- **Base Employee Management System SRS Document 1** (Current submission)
- **Term I Assignment Implementation Report** (DivyanshDBSA.pdf, Section 8.2, p. 34)
- **IEEE 830–1998, ISO/IEC/IEEE 29148:2018 -** Systems and Software Engineering Requirements
- **SQLite BLOB Documentation -** Official SQLite3 Binary Data Reference
- **React.js Documentation -** Component Integration Guidelines

---

2. Overall Description

2.1 Product Functions Extension

The Employee Profile Photo Extension integrates with the existing Employee Management Database System developed using React.js, Node.js, and SQLite3. It extends the base system's capabilities without modifying core functionality.

New capabilities include:

- Upload, store, retrieve, and display employee profile photos
- File type, size, and security validation for image uploads
- Visual employee identification through profile photos
- Efficient BLOB handling within SQLite3 architecture
- **Integration with existing Employee detail views and listing pages**

2.2 User Interaction Model

User Role	Image Module Access	Operations Permitted
Database Administrator	System-level image storage management and optimization	Full schema modification, backup operations including BLOB data
HR Manager	Full CRUD access to employee profile photos	Upload, update, delete employee photos; bulk photo management
Project Manager	Read-only access to view employee photos in project assignments	View photos in project team listings and reports
Employee	View personal profile photo; request photo updates	Personal photo viewing; update requests through HR

2.3 Integration Strategy

The extension follows the **principle of minimal disruption** to existing system architecture:

- **Database Level:** Addition of single BLOB column to existing EMPLOYEE table
- **API Level:** New RESTful endpoints for image operations alongside existing employee endpoints
- **Frontend Level:** Enhancement of existing React components with image display and upload capabilities
- **Security Level:** Extension of existing validation framework to include file security checks

2.4 Constraints and Dependencies

- **Schema Constraint:** Must maintain BCNF normalization of base Employee Management schema
- **Technology Constraint:** Must integrate seamlessly with existing React.js + Node.js + SQLite3 stack
- **Performance Constraint:** Image operations must not degrade existing system performance
- **Security Constraint:** Image storage must follow same security principles as base system
- **Backward Compatibility:** All existing Employee Management operations must remain functional

---

3. Specific Requirements (Image Extension)

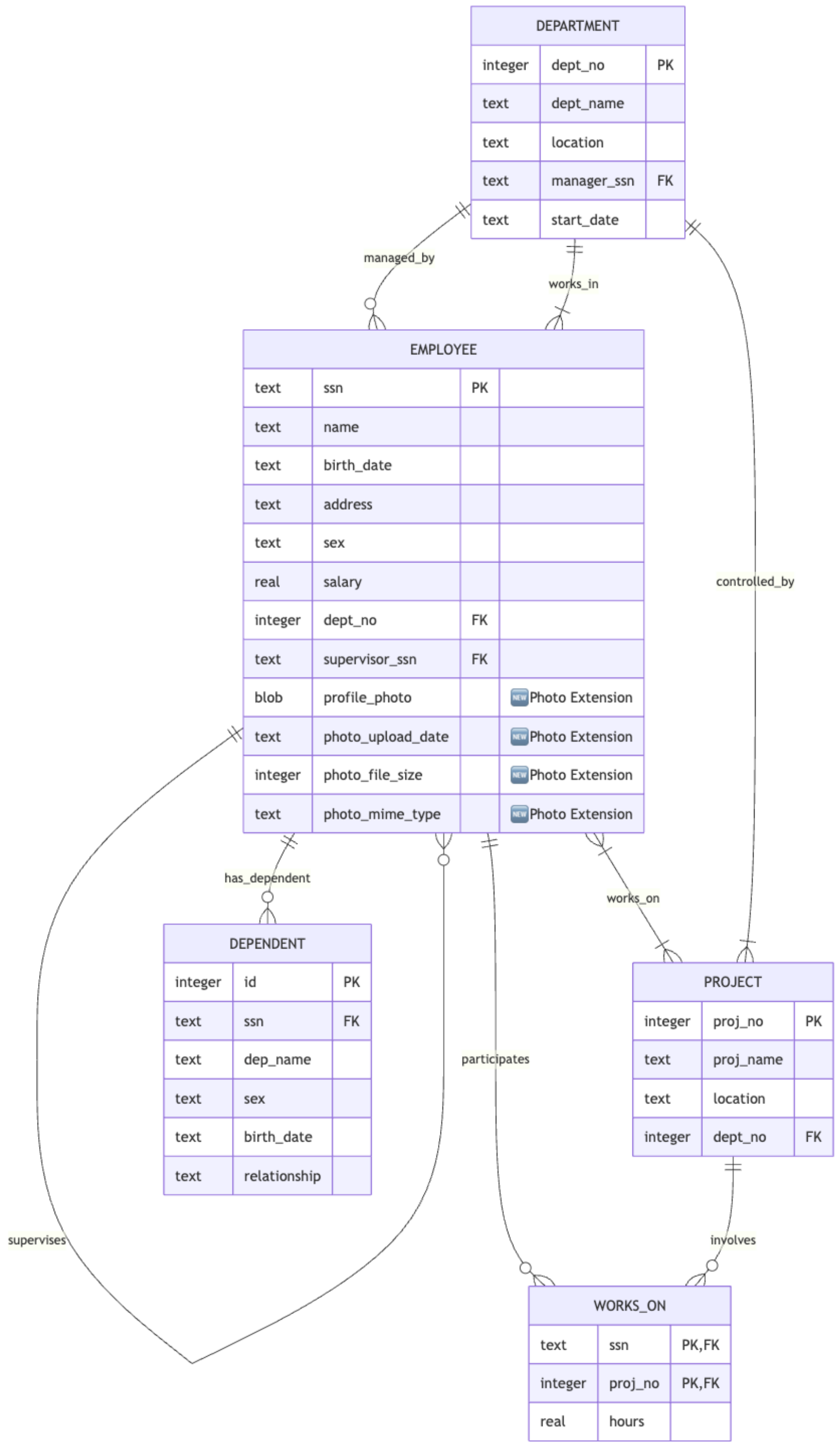
3.1 Functional Requirements (FR-M)

ID	Requirement Description	Priority
FR-M1	The system shall allow authorized users to upload an Employee profile photo.	High
FR-M2	The system shall validate uploaded files to ensure they are of type JPEG or PNG.	High
FR-M3	The system shall store the uploaded photo as a BLOB in the EMPLOYEE table.	High
FR-M4	The system shall allow updating/replacing an existing Employee profile photo.	Medium
FR-M5	The system shall allow displaying the stored profile photo on the Employee detail view.	High
FR-M6	The system shall prevent upload of files larger than the configured size limit (5 MB).	High
FR-M7	The backend shall sanitize file names and prevent arbitrary file access.	High
FR-M8	The system shall provide fallback placeholder images when an Employee has no photo uploaded.	Medium

3.2 Non-Functional Requirements (NFR-M)

ID	Category	Requirement Description	Priority
NFR-M1	Performance	Image loading should not exceed 1 second on a standard network.	High
NFR-M2	Performance	Images shall be compressed (if stored as files) to optimize load time.	Medium
NFR-M3	Security	The system shall not expose direct file paths; signed URLs or APIs shall be used.	High
NFR-M4	Security	Uploaded files must be scanned/validated for MIME correctness.	High
NFR-M5	Domain	Maximum supported file size is 5 MB.	High
NFR-M6	Domain	Supported formats are PNG and JPEG only.	High
NFR-M7	Maintainability	The addition of BLOBs must not require changes to existing CRUD operations.	Medium
NFR-M8	Scalability	The schema should allow migration to cloud storage (AWS S3/GCP) without breaking changes.	Medium

4. Extended Database Design



4.1 Extended EMPLOYEE Table DDL

Primary Extension: Addition of Profile_Photo BLOB column to existing normalized schema

```sql
-- Extension DDL for Employee Management Database
-- Adds multimodal image storage capability while maintaining BCNF

ALTER TABLE EMPLOYEE
ADD COLUMN Profile_Photo BLOB;

-- Optional: Add metadata columns for enhanced image management
-- ALTER TABLE EMPLOYEE
-- ADD COLUMN Photo_Upload_Date DATETIME DEFAULT CURRENT_TIMESTAMP;
-- ALTER TABLE EMPLOYEE
-- ADD COLUMN Photo_File_Size INTEGER;
-- ALTER TABLE EMPLOYEE
-- ADD COLUMN Photo_MIME_Type VARCHAR(50);
```

4.2 Schema Integration and BCNF Compliance

The image extension maintains **full BCNF compliance** of the existing Employee Management schema:

```sql
-- Complete Extended EMPLOYEE Schema (BCNF Maintained)
EMPLOYEE (
    SSN VARCHAR(11) PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Birth_Date DATE,
    Address VARCHAR(200),
    Sex CHAR(1) CHECK(Sex IN ('M','F')),
    Salary DECIMAL(10,2) CHECK(Salary > 0),
    Dept_No INTEGER REFERENCES DEPARTMENT(Dnumber),
    Supervisor_SSN VARCHAR(11) REFERENCES EMPLOYEE(SSN),
    Profile_Photo BLOB  -- NEW: Image storage capability
);
```

BCNF Verification:

- Profile_Photo depends solely on SSN (primary key)
- No new functional dependencies introduced
- All existing dependencies remain valid
- Extension preserves all referential integrity constraints

4.3 Storage Strategy and Implementation

Approach	Implementation	Advantages	Trade-offs	Selected
BLOB Storage	Direct binary storage in SQLite	Simple backup, transaction consistency, referential integrity	Database size growth, memory usage	✅ **YES**
File Path Storage	Store file paths, images in filesystem	Smaller database size, easier file management	Complex backup procedures, broken referential integrity	❌ No
Hybrid Approach	Small images in BLOB, large images as files	Balanced performance and storage	Increased complexity	❌ No

Selected Approach Justification: BLOB Storage demonstrates the multimodal BLOB provisions from Term I design and maintains the integrity principles of the normalized database system.

4.4 Multimodal Data Handling

Data Type Specifications:

- **Storage Format:** SQLite BLOB (Binary Large Object)
- **Maximum Size:** 5 MB per image (application-enforced)
- **Supported Formats:** JPEG, PNG (MIME validation enforced)
- **Compression:** Client-side compression recommended before upload
- **Retrieval Method:** Base64 encoding for web display or direct binary transfer

5. System Integration

5.1 API Endpoints

Endpoint	Method	Function	Request Body	Response	Status Codes
`/api/employees/:id/photo`	**GET**	Retrieve employee photo	None	Binary image data or placeholder	200, 404, 500
`/api/employees/:id/photo`	**POST**	Upload new profile photo	multipart/form-data with image file	`{"success": true, "message": "Photo uploaded"}`	201, 400, 413, 422
`/api/employees/:id/photo`	**PUT**	Update existing photo	multipart/form-data with image file	`{"success": true, "message": "Photo updated"}`	200, 400, 404, 413
`/api/employees/:id/photo`	**DELETE**	Remove profile photo	None	`{"success": true, "message": "Photo deleted"}`	200, 404, 500

API Integration with Base System:

- Photo endpoints extend existing `/api/employees` API structure
- Consistent authentication and authorization with base employee endpoints
- Error handling follows same patterns as base system
- Request/response format maintains API consistency

5.2 Frontend Integration

React Component Extensions:

Component	Enhancement	Integration Point
EmployeeCard	Display thumbnail profile photos (64x64px)	Employee listing pages, search results
EmployeeDetails	Full-size photo display (200x200px) with upload interface	Employee detail view, profile management
PhotoUpload	Drag-and-drop image upload with preview and validation	Employee creation/edit forms
ImageValidator	Client-side file validation (type, size) before upload	All photo upload interfaces
PhotoPlaceholder	Default avatar display when no photo exists	All employee display contexts

UI Integration Strategy:

- **Non-intrusive Design:** Photo features integrate seamlessly without disrupting existing UI flow
- **Progressive Enhancement:** System functions normally even if image features fail
- **Responsive Design:** Photo displays adapt to mobile and desktop viewports
- **Accessibility:** Alt text and screen reader support for all image elements

5.3 Backend Integration

Node.js Integration Points:

```
// Example integration with existing Employee controller
const employeeController = {
  // Existing methods remain unchanged
  getEmployee: async (req, res) => {
    /* existing code */
  },
  createEmployee: async (req, res) => {
    /* existing code */
  },
  updateEmployee: async (req, res) => {
    /* existing code */
  },

  // New photo-specific methods
  getEmployeePhoto: async (req, res) => {
    // BLOB retrieval and binary response
  },
  uploadEmployeePhoto: async (req, res) => {
    // File validation, BLOB storage
  },
};
```

Database Integration:

- **Connection Reuse:** Photo operations use existing SQLite connection pool
- **Transaction Management:** Photo uploads participate in existing transaction framework
- **Error Handling:** Photo errors integrate with existing error middleware
- **Logging:** Photo operations logged through existing system logger

---

6. Validation and Verification (Image Extension)

6.1 Image Module Test Cases

Test ID	Test Category	Test Description	Input	Expected Result	Priority
T-M1	File Size Validation	Upload file >5MB	6 MB JPEG image	Upload rejected with size limit error	High
T-M2	File Type Validation	Upload invalid format	PDF file disguised as image	Upload rejected with format error	High
T-M3	Successful Upload	Upload valid image	200 KB JPEG image	Image stored successfully in BLOB	High
T-M4	Image Retrieval	Retrieve existing photo	GET /api/employees/123/photo	Image displays correctly	High
T-M5	Missing Image	Retrieve non-existent photo	GET /api/employees/999/photo	Placeholder image returned	Medium
T-M6	Photo Replacement	Replace existing image	Upload new image for existing employee	Old BLOB replaced with new image	Medium
T-M7	Security Validation	MIME type spoofing attempt	Malicious file with image extension	Upload blocked by MIME validation	High
T-M8	Performance Test	Load employee list with photos	50 employees with profile photos	All images load within 1-second requirement	Medium

6.2 Integration Test Cases

Test ID	Test Description	Expected Result
T-I1	Existing CRUD operations remain functional after BLOB addition	All original functionality unaffected
T-I2	Database backup includes BLOB data	Complete system restore with images
T-I3	Employee deletion cascades properly with image data	BLOB data cleaned up with employee record

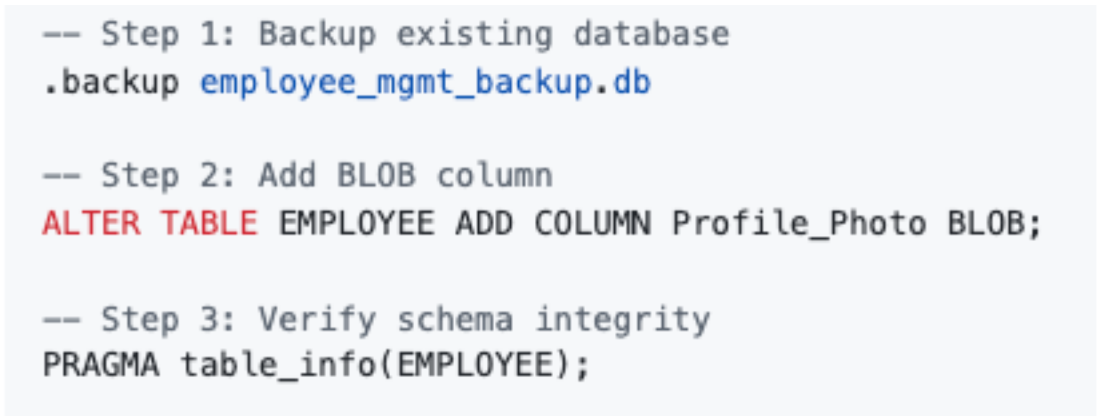
7. Implementation Guidelines

7.1 Development Phases

Phase	Deliverable	Timeline	Dependencies	Success Criteria
Phase 1	Extended DDL and BLOB storage implementation	Week 1	Base Employee schema validated	EMPLOYEE table successfully extended, BCNF maintained
Phase 2	Backend API development for image operations	Week 2	Phase 1 complete, Node.js environment ready	All photo endpoints functional, validation working
Phase 3	Frontend integration and upload interface	Week 3	Phase 2 complete, React.js environment ready	Photo upload/display components integrated
Phase 4	Testing, validation, and performance optimization	Week 4	Phases 1-3 complete	All test cases pass, performance targets met

7.2 Technical Implementation

Phase 1: Database Extension



```sql
-- Step 1: Backup existing database
.backup employee_mgmt_backup.db

-- Step 2: Add BLOB column
ALTER TABLE EMPLOYEE ADD COLUMN Profile_Photo BLOB;

-- Step 3: Verify schema integrity
PRAGMA table_info(EMPLOYEE);
```

Phase 2: Backend Implementation

```
// Image upload middleware with validation
const multer = require("multer");
const upload = multer({
  limits: { fileSize: 5 * 1024 * 1024 }, // 5MB limit
  fileFilter: (req, file, cb) => {
    const allowedTypes = ["image/jpeg", "image/png"];
    cb(null, allowedTypes.includes(file.mimetype));
  },
});

// Photo upload endpoint
app.post(
  "/api/employees/:id/photo",
  authenticate,
  upload.single("photo"),
  async (req, res) => {
    try {
      const { id } = req.params;
      const imageBuffer = req.file.buffer;

      // Store BLOB in database
      await db.run("UPDATE EMPLOYEE SET Profile_Photo = ? WHERE SSN = ?", [
        imageBuffer,
        id,
      ]);

      res.status(201).json({
        success: true,
        message: "Photo uploaded successfully",
      });
    } catch (error) {
      res.status(500).json({ error: error.message });
    }
  }
);
```

Phase 3: Frontend Implementation

```
// Photo upload component
const PhotoUpload = ({ employeeId, onUploadSuccess }) => {
  const [uploading, setUploading] = useState(false);

  const handleFileUpload = async (file) => {
    setUploading(true);
    const formData = new FormData();
    formData.append("photo", file);

    try {
      const response = await fetch(`/api/employees/${employeeId}/photo`, {
        method: "POST",
        body: formData,
        headers: {
          Authorization: `Bearer ${localStorage.getItem("token")}`,
        },
      });

      if (response.ok) {
        onUploadSuccess();
      }
    } catch (error) {
      console.error("Upload failed:", error);
    } finally {
      setUploading(false);
    }
  };

  return (
    <div className="photo-upload">
      <input
        type="file"
        accept="image/jpeg,image/png"
        onChange={(e) => handleFileUpload(e.target.files[0])}
        disabled={uploading}
      />
      {uploading && <div>Uploading...</div>}
    </div>
  );
};
```

7.3 Success Metrics

Metric	Target	Measurement Method	Acceptance Criteria
Upload Success Rate	>99% for valid files	Automated testing with sample images	All valid JPEG/PNG files under 5MB upload successfully
Image Load Time	<1 second average	Performance monitoring in browser	Photo display completes within 1s on standard connection
Storage Efficiency	<5MB average per employee	Database size monitoring	Photo storage remains within size constraints
System Performance	No degradation in existing operations	Before/after performance comparison	Employee CRUD operations maintain original response times
User Satisfaction	>90% positive feedback	User acceptance testing	Users find photo features intuitive and helpful

8. Appendix

8.1 Future Multimodal Enhancements

Immediate Extensions (Next Phase):

- **Document Storage:** Extend BLOB capability to store employee documents (resumes, certificates)
- **Spatial Data Integration:** Add geographic coordinates for employee work locations
- **Audio Notes:** Voice annotations for employee profiles
- **Video Profiles:** Short video introductions for remote team members

Advanced Multimodal Features:

- **AI-Powered Image Analysis:** Automatic photo quality assessment and optimization
- **Facial Recognition Integration:** Secure access control using employee photos
- **Mobile Photo Capture:** Direct upload from mobile devices with camera integration
- **Bulk Photo Import:** Mass upload capabilities for HR onboarding processes

System Evolution Path:

- **Cloud Storage Migration:** Transition from BLOB to cloud storage (AWS S3, Azure Blob) while maintaining API consistency
- **Content Delivery Network:** Implement CDN for global photo access optimization
- **Image Processing Pipeline:** Automated resizing, format conversion, and compression
- **Backup and Archival:** Specialized backup strategies for large multimodal datasets

8.2 Document History

Version	Date	Description	Author
1.0	**Nov 2025**	Initial SRS for Employee Profile Photo Extension - BLOB implementation demonstrating multimodal database capabilities	Divyansh

---

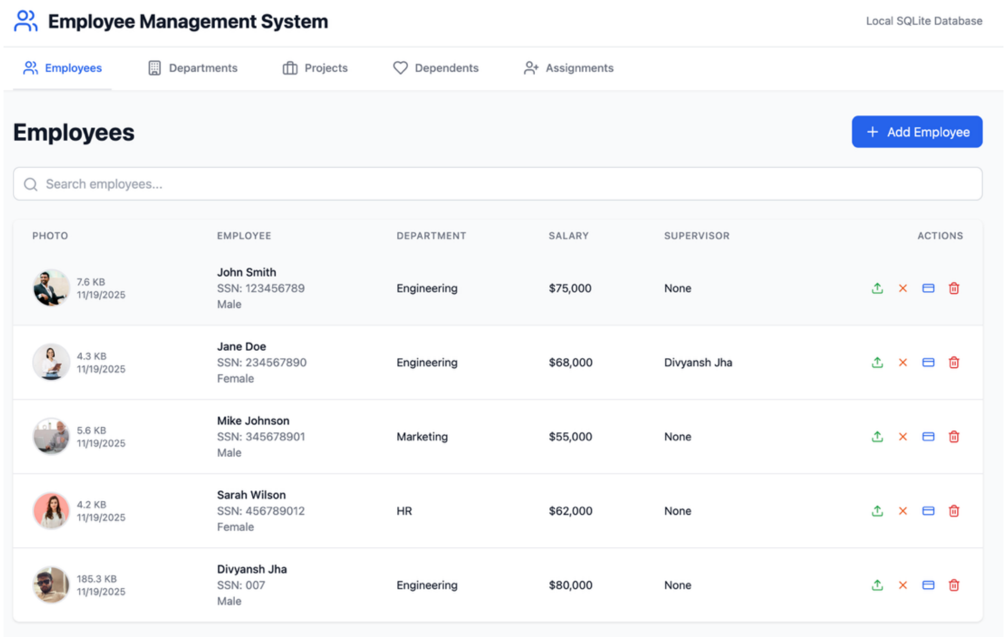
Compliance Statement:
This multimodal extension SRS complies with **IEEE 830 and ISO/IEC/IEEE 29148**, including:

- **Complete requirement specification** for multimodal database extension
- **Full integration documentation** with base Employee Management System
- **Comprehensive testing strategy** for BLOB data handling
- **Implementation roadmap** with measurable success criteria
- **Future extensibility planning** for additional multimodal capabilities

Technical Achievement:
This extension successfully demonstrates the **practical implementation of multimodal database provisions** designed in Term I, validating the forward-thinking architecture of the base Employee Management System and establishing a foundation for advanced database capabilities.

---

9. Screenshots:

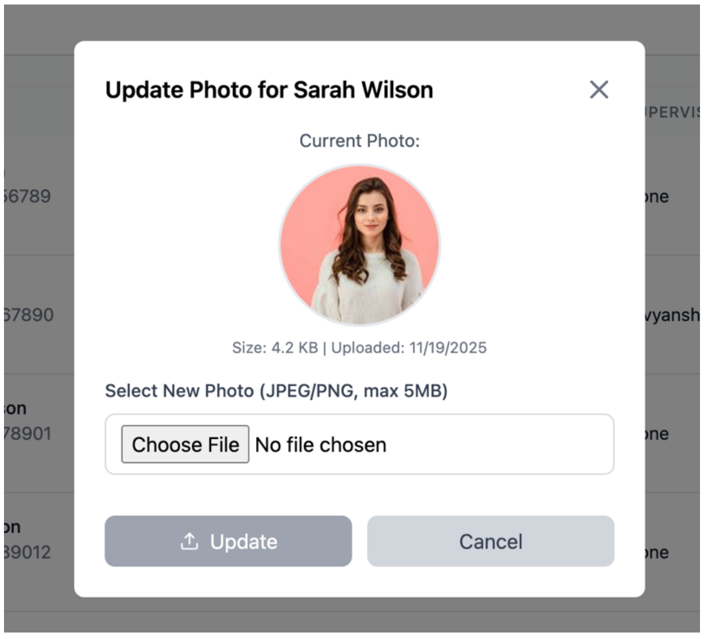


Employee Management System

Local SQLite Database

Employees

PHOTO	EMPLOYEE	DEPARTMENT	SALARY	SUPERVISOR	ACTIONS
7.6 KB 11/19/2025	John Smith SSN: 123456789 Male	Engineering	$75,000	None	
4.3 KB 11/19/2025	Jane Doe SSN: 234567890 Female	Engineering	$68,000	Divyansh Jha	
5.6 KB 11/19/2025	Mike Johnson SSN: 345678901 Male	Marketing	$55,000	None	
4.2 KB 11/19/2025	Sarah Wilson SSN: 456789012 Female	HR	$62,000	None	
185.3 KB 11/19/2025	Divyansh Jha SSN: 007 Male	Engineering	$80,000	None	



Update Photo for Sarah Wilson

Current Photo:

Size: 4.2 KB | Uploaded: 11/19/2025

Select New Photo (JPEG/PNG, max 5MB)