BIKA - Document Filling System

Technical Documentation

Table of Contents

Project Overview

Mission Statement

BIKA is a modern, scalable document management system designed to help companies and individuals efficiently archive, organize, and retrieve both digital and physical documents while maintaining clear traceability of physical storage locations.

Core Objectives

- **Digital-Physical Integration**: Bridge the gap between digital archives and physical document storage
- **Hierarchical Organization**: Support multi-level organizational structures (Companies → Departments → Users)
- **Flexible Document Types**: Allow customizable document types with configurable metadata fields
- **Advanced Search Capabilities**: Provide powerful search functionality across all document attributes
- **Personal Drive Storage**: Offer individual storage space with configurable limits
- **Location Tracking**: Maintain precise physical location data (Office → Cupboard → Drawer → Color-coded folders)

Technology Stack

Frontend

- **Framework**: React 18+ with TypeScript

- **State Management**: Redux Toolkit / Zustand
- **UI Library**: Material-UI v5 / Ant Design
- **Build Tool**: Vite
- **Testing**: Jest + React Testing Library

Backend

- **Framework**: Spring Boot 3.x
- **Language**: Java 17+
- **Database**: PostgreSQL 14+
- **Authentication**: Spring Security + JWT
- **File Storage**: AWS S3 / MinIO
- **Search Engine**: Elasticsearch
- **API Documentation**: OpenAPI 3.0 (Swagger)

Infrastructure

- **Containerization**: Docker + Docker Compose
- **Orchestration**: Kubernetes
- **CI/CD**: GitHub Actions / Jenkins
- **Monitoring**: Spring Boot Actuator + Prometheus + Grafana
- **Caching**: Redis

Use Cases

Primary Use Cases

UC-001: Company Registration & Management

Actor: BIKA Super Administrator **Description**: Manage company onboarding and approval process **Flow**:

1. Company requests access to BIKA system
2. Super admin reviews and approves/rejects request
3. Upon approval, super admin creates company profile
4. Super admin assigns company administrator

UC-002: User Management

Actor: Company Administrator **Description**: Manage company users and department structure **Flow**:

1. Create and manage departments
2. Add users to specific departments
3. Assign roles and permissions
4. Manage user access levels

UC-003: Document Type Configuration

Actor: Company Administrator / Department Manager **Description**: Define and configure document types with custom fields **Flow**:

1. Create new document type
2. Define required and optional metadata fields
3. Set field validation rules
4. Assign document type to departments
5. Configure access permissions

UC-004: Document Archival

Actor: Company User **Description**: Archive documents with complete metadata and physical location **Flow**:

1. Navigate to specific document type
2. Create folder structure (if needed)
3. Create new document entry
4. Fill in configured metadata fields
5. Upload digital document
6. Specify physical location (Office, Cupboard, Drawer, Color)
7. Save document entry

UC-005: Advanced Document Search

Actor: Company User **Description**: Search for documents using multiple criteria **Flow**:

1. Access search interface
2. Apply filters (document type, date range, metadata, location)
3. Execute search query
4. View search results with location information
5. Access document details and files

UC-006: Personal Drive Management

Actor: Company User **Description**: Manage personal file storage with size limitations **Flow**:

1. Access personal drive
2. Upload files (within 2GB limit)
3. Organize files in folders
4. Share files with other users (if permitted)
5. Monitor storage usage

System Roles & Permissions

Role Hierarchy

1. BIKA Super Administrator

Scope: System-wide **Permissions**:

- Manage all companies
- Approve/reject company registration requests
- Assign company administrators
- System-wide configuration and monitoring
- Access to all system analytics and reports

2. Company Administrator

Scope: Single company **Permissions**:

- Manage company profile and settings
- Create and manage departments
- Add/remove company users
- Assign department managers
- Configure document types (company-wide)
- Access company-wide reports and analytics
- Manage storage quotas and limits

3. Department Manager

Scope: Single department within company **Permissions**:

- Manage department users
- Configure document types for department
- Create and configure folder structures
- Access department reports
- Manage department-specific settings
- Approve/reject document access requests

4. Company User (Employee)

Scope: Assigned departments within company **Permissions**:

- Archive documents in accessible document types
- Search and view accessible documents
- Manage personal drive (2GB limit)
- Create folder structures within permitted areas
- Update document metadata (if permitted)

5. Guest User (Read-Only)

Scope: Limited document access within company **Permissions**:

- View specific documents (as granted)
- Search within permitted document types
- Download permitted documents

- No upload or modification capabilities

Permission Matrix

Feature	Super Admin	Company Admin	Dept Manager	User	Guest
Company Management	✅	❌	❌	❌	❌
User Management	✅	✅	✅*	❌	❌
Document Type Config	✅	✅	✅*	❌	❌
Document Archive	✅	✅	✅	✅	❌
Document Search	✅	✅	✅	✅	✅*
Personal Drive	❌	✅	✅	✅	❌
System Reports	✅	✅*	✅*	❌	❌

*Limited to their scope

Backend Architecture

Architectural Pattern

Layered Architecture with **Domain-Driven Design (DDD)** principles

Project Structure

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── bika/
│   │           ├── BikaApplication.java
│   │           ├── config/           # Configuration classes
│   │           ├── controller/       # REST Controllers
│   │           ├── service/          # Business Logic Layer
│   │           ├── repository/       # Data Access Layer
│   │           ├── entity/           # JPA Entities
│   │           ├── dto/              # Data Transfer Objects
│   │           ├── security/         # Security configuration
│   │           ├── exception/        # Custom exceptions
│   │           ├── util/             # Utility classes
│   │           └── constants/        # Application constants
│   └── resources/
│       ├── application.yml
│       ├── application-dev.yml
│       ├── application-prod.yml
│       └── db/migration/           # Flyway migrations
```

Core Modules

1. Authentication & Authorization Module

- JWT-based authentication

- Role-based access control (RBAC)
- OAuth2 integration capability
- Session management
- Password policies and encryption

2. Company Management Module

- Company registration and approval workflow
- Multi-tenancy support
- Company-specific configurations
- Department hierarchy management

3. User Management Module

- User lifecycle management
- Role assignment and permissions
- Department associations
- User profile management

4. Document Management Module

- Document type configuration
- Metadata schema management
- File upload and storage
- Folder structure management
- Physical location tracking

5. Search Module

- Elasticsearch integration
- Advanced search capabilities
- Indexing strategies
- Search result ranking and filtering

6. Personal Drive Module

- Individual file storage
- Storage quota management
- File sharing capabilities
- Access control for personal files

7. Reporting Module

- Document usage analytics
- Storage utilization reports
- User activity reports
- System performance metrics

API Specifications

API Design Principles

- **RESTful Architecture**: Following REST conventions
- **Consistent Response Format**: Standardized JSON responses
- **Versioning**: URI versioning (e.g., `/api/v1/`)
- **Pagination**: Cursor-based pagination for large datasets
- **Rate Limiting**: To prevent abuse and ensure fair usage
- **HATEOAS**: Hypermedia-driven API responses where applicable

Authentication Endpoints

POST `/api/v1/auth/login`

```
Request:
{
  "email": "user@company.com",
  "password": "securePassword",
  "companyId": "company-uuid"
}

Response:
{
  "success": true,
  "data": {
    "accessToken": "jwt-token",
    "refreshToken": "refresh-token",
    "user": {
      "id": "user-uuid",
      "name": "John Doe",
      "email": "user@company.com",
      "role": "COMPANY_USER",
      "permissions": ["DOCUMENT_READ", "DOCUMENT_WRITE"]
    }
  },
  "timestamp": "2024-01-01T00:00:00Z"
}
```

POST `/api/v1/auth/refresh`

```
Request:
{
  "refreshToken": "refresh-token"
}

Response:
{
  "success": true,
  "data": {
    "accessToken": "new-jwt-token",
    "refreshToken": "new-refresh-token"
  }
}
```

Company Management Endpoints

POST `/api/v1/companies`

Role Required: BIKA_SUPER_ADMIN

```
Request:
{
  "name": "Acme Corporation",
  "email": "admin@acme.com",
  "phone": "+1234567890",
  "address": {
    "street": "123 Business Ave",
    "city": "Business City",
    "state": "BC",
    "zipCode": "12345",
    "country": "USA"
  },
  "adminUser": {
    "name": "Jane Admin",
    "email": "jane@acme.com",
    "phone": "+1234567891"
  }
}

Response:
{
  "success": true,
  "data": {
    "id": "company-uuid",
    "name": "Acme Corporation",
    "status": "ACTIVE",
    "createdAt": "2024-01-01T00:00:00Z"
  }
}
```

GET `/api/v1/companies/{companyId}/departments`

Role Required: COMPANY_ADMIN, DEPT_MANAGER

```
Response:
{
  "success": true,
  "data": {
    "departments": [
      {
        "id": "dept-uuid",
        "name": "Human Resources",
        "description": "HR Department",
        "managerCount": 2,
        "userCount": 15,
        "createdAt": "2024-01-01T00:00:00Z"
      }
    ],
    "pagination": {
      "page": 1,
      "size": 20,
      "total": 5,
```

```
      "hasNext": false
    }
  }
}
```

Document Management Endpoints

POST `/api/v1/document-types`

Role Required: COMPANY_ADMIN, DEPT_MANAGER

```
Request:
{
  "name": "Employee Contract",
  "description": "Employment contract documents",
  "departmentIds": ["dept-uuid-1", "dept-uuid-2"],
  "fields": [
    {
      "name": "employeeName",
      "label": "Employee Name",
      "type": "TEXT",
      "required": true,
      "validation": {
        "minLength": 2,
        "maxLength": 100
      }
    },
    {
      "name": "contractDate",
      "label": "Contract Date",
      "type": "DATE",
      "required": true
    },
    {
      "name": "salary",
      "label": "Annual Salary",
      "type": "NUMBER",
      "required": false,
      "validation": {
        "min": 0,
        "max": 1000000
      }
    }
  ]
}

Response:
{
  "success": true,
  "data": {
    "id": "doc-type-uuid",
    "name": "Employee Contract",
    "status": "ACTIVE",
    "fieldCount": 3,
    "createdAt": "2024-01-01T00:00:00Z"
  }
}
```

POST `/api/v1/documents`

Role Required: COMPANY_USER (with appropriate department access)

```
Request (multipart/form-data):
{
  "documentTypeId": "doc-type-uuid",
  "folderId": "folder-uuid",
  "metadata": {
    "employeeName": "John Smith",
    "contractDate": "2024-01-15",
    "salary": 75000
  },
  "physicalLocation": {
    "office": "Main Office",
    "cupboard": "Cabinet A",
    "drawer": "Drawer 3",
    "folderColor": "Blue"
  },
  "file": [binary-data]
}

Response:
{
  "success": true,
  "data": {
    "id": "document-uuid",
    "documentTypeId": "doc-type-uuid",
    "fileName": "contract_john_smith.pdf",
    "fileSize": 1024000,
    "uploadedAt": "2024-01-01T00:00:00Z",
    "physicalLocation": {
      "office": "Main Office",
      "cupboard": "Cabinet A",
      "drawer": "Drawer 3",
      "folderColor": "Blue"
    }
  }
}
```

GET `/api/v1/documents/search`

Role Required: COMPANY_USER

```
Request (Query Parameters):
?q=john smith
&documentTypeId=doc-type-uuid
&dateFrom=2024-01-01
&dateTo=2024-12-31
&office=Main Office
&page=1
&size=20

Response:
{
  "success": true,
  "data": {
    "documents": [
      {
        "id": "document-uuid",
        "documentType": "Employee Contract",
```

```
        "fileName": "contract_john_smith.pdf",
        "metadata": {
          "employeeName": "John Smith",
          "contractDate": "2024-01-15"
        },
        "physicalLocation": {
          "office": "Main Office",
          "cupboard": "Cabinet A",
          "drawer": "Drawer 3",
          "folderColor": "Blue"
        },
        "uploadedAt": "2024-01-01T00:00:00Z",
        "uploadedBy": "Jane Doe"
      }
    ],
    "pagination": {
      "page": 1,
      "size": 20,
      "total": 150,
      "hasNext": true
    },
    "aggregations": {
      "documentTypes": {
        "Employee Contract": 45,
        "Invoice": 78,
        "Policy Document": 27
      },
      "offices": {
        "Main Office": 120,
        "Branch Office": 30
      }
    }
  }
}
```

Personal Drive Endpoints

POST `/api/v1/drive/files`

Role Required: COMPANY_USER

```
Request (multipart/form-data):
{
  "folderId": "drive-folder-uuid", // optional
  "file": [binary-data],
  "description": "Project presentation slides"
}

Response:
{
  "success": true,
  "data": {
    "id": "drive-file-uuid",
    "fileName": "presentation.pptx",
    "fileSize": 5242880,
    "mimeType":
"application/vnd.openxmlformats-officedocument.presentationml.presentation",
    "uploadedAt": "2024-01-01T00:00:00Z",
    "storageUsed": "1.2GB",
```

```
    "storageLimit": "2GB"
  }
}
```

GET `/api/v1/drive/usage`

Role Required: COMPANY_USER

```
Response:
{
  "success": true,
  "data": {
    "used": 1288490188,
    "limit": 2147483648,
    "usagePercentage": 60.0,
    "fileCount": 47,
    "folderCount": 8
  }
}
```

Folder Management Endpoints

POST /api/v1/folders

```
{
  "name": "HR Documents 2024",
  "description": "Human Resources documents for 2024",
  "parentFolderId": "parent-folder-uuid"
}
```

GET /api/v1/folders/{folderId}/contents

```
{
  "success": true,
  "data": {
    "folders": [
      {
        "id": "folder-uuid",
        "name": "Contracts",
        "documentCount": 25,
        "subfolderCount": 3,
        "createdAt": "2024-01-01T00:00:00Z"
      }
    ],
    "documents": [
      {
        "id": "doc-uuid",
        "fileName": "contract.pdf",
        "documentType": "Employment Contract",
        "physicalLocation": {
          "office": "Main Office",
          "cupboard": "Cabinet A",
          "drawer": "Drawer 1",
          "folderColor": "Blue"
        }
      }
    ]
  }
}
```

Advanced Search Endpoints

GET /api/v1/documents/advanced-search

```
Query Parameters:
- q: string (search query)
- documentTypeIds: array of UUIDs
- folderIds: array of UUIDs
- dateFrom: ISO date
- dateTo: ISO date
- office: string
- cupboard: string
- drawer: string
- folderColor: string
- metadata: JSON object for metadata filters
- page: integer
- size: integer
- sortBy: string (fileName, createdAt, etc.)
- sortOrder: string (ASC, DESC)
```

User Management Endpoints

POST /api/v1/users

```
{
  "email": "john.doe@company.com",
  "firstName": "John",
  "lastName": "Doe",
  "phone": "+1234567890",
  "role": "COMPANY_USER",
  "departmentIds": ["dept-uuid-1", "dept-uuid-2"],
  "temporaryPassword": "TempPass123!"
}
```

PUT /api/v1/users/{userId}/departments

```
{
  "departmentIds": ["dept-uuid-1", "dept-uuid-2"],
  "roles": {
    "dept-uuid-1": "MEMBER",
    "dept-uuid-2": "MANAGER"
  }
}
```

Security Implementation

Authentication Strategy

JWT (JSON Web Tokens)

- **Access Token**: Short-lived (15 minutes) for API access
- **Refresh Token**: Long-lived (7 days) for token renewal
- **Token Storage**: HttpOnly cookies for web clients
- **Token Rotation**: Automatic refresh token rotation

Multi-Factor Authentication (MFA)

- **TOTP Support**: Time-based One-Time Password
- **SMS Backup**: SMS-based verification as fallback
- **Recovery Codes**: One-time use backup codes

Authorization Framework

Role-Based Access Control (RBAC)

```
@PreAuthorize("hasRole('COMPANY_ADMIN') or hasRole('DEPT_MANAGER')")
public DocumentType createDocumentType(CreateDocumentTypeRequest request) {
    // Implementation
}

@PreAuthorize("@documentSecurityService.canAccessDocument(#documentId,
authentication)")
public Document getDocument(@PathVariable String documentId) {
    // Implementation
}
```

Resource-Based Permissions

- **Document Access**: Based on department membership and document type permissions
- **Folder Access**: Hierarchical permissions with inheritance
- **Company Isolation**: Strict tenant separation

Data Protection

Encryption

- **At Rest**: AES-256 encryption for sensitive data
- **In Transit**: TLS 1.3 for all communications
- **Database**: Column-level encryption for PII
- **File Storage**: Server-side encryption with customer-managed keys

Data Privacy

- **GDPR Compliance**: Right to erasure, data portability, consent management
- **Data Minimization**: Collect only necessary information
- **Audit Logging**: Comprehensive activity logging
- **Data Retention**: Configurable retention policies

Security Headers

```
security:
  headers:
    content-security-policy: "default-src 'self'; script-src 'self'
'unsafe-inline'"
    x-frame-options: "DENY"
    x-content-type-options: "nosniff"
    x-xss-protection: "1; mode=block"
    strict-transport-security: "max-age=31536000; includeSubDomains"
```

Rate Limiting

```
rate-limiting:
  global:
    requests-per-minute: 1000
```

```
  authentication:
    requests-per-minute: 10
    burst: 5
  file-upload:
    requests-per-minute: 30
    max-file-size: 100MB
```

Advanced Authentication Features

```
@Component
public class JwtTokenProvider {

    private final String secretKey;
    private final long accessTokenValidityInMs;
    private final long refreshTokenValidityInMs;

    public String createAccessToken(Authentication authentication) {
        UserPrincipal userPrincipal = (UserPrincipal)
authentication.getPrincipal();
        Date expiryDate = new Date(System.currentTimeMillis() +
accessTokenValidityInMs);

        return Jwts.builder()
                .setSubject(userPrincipal.getId())
                .setIssuedAt(new Date())
                .setExpiration(expiryDate)
                .claim("companyId", userPrincipal.getCompanyId())
                .claim("role", userPrincipal.getRole())
                .claim("permissions", userPrincipal.getPermissions())
                .signWith(SignatureAlgorithm.HS512, secretKey)
                .compact();
    }

    public String createRefreshToken(String userId) {
        Date expiryDate = new Date(System.currentTimeMillis() +
refreshTokenValidityInMs);

        return Jwts.builder()
                .setSubject(userId)
                .setIssuedAt(new Date())
                .setExpiration(expiryDate)
                .claim("type", "refresh")
                .signWith(SignatureAlgorithm.HS512, secretKey)
                .compact();
    }
}
```

Multi-Factor Authentication

```
@Service
public class MfaService {

    private final GoogleAuthenticator googleAuthenticator;

    public String generateSecretKey(String userId) {
        final GoogleAuthenticatorKey key = googleAuthenticator.createKey();
        // Store key securely in database
        return key.getKey();
    }

    public boolean validateTotp(String userId, int code) {
        String secretKey = getUserSecretKey(userId);
```

```
        return googleAuthenticator.authorize(secretKey, code);
    }

    public String generateQrCodeUrl(String userId, String companyName) {
        String secretKey = getUserSecretKey(userId);
        return GoogleAuthenticatorQRGenerator.getOtpAuthURL(
            "BIKA",
            userId + "@" + companyName,
            new GoogleAuthenticatorKey.Builder(secretKey).build()
        );
    }
}
```

File Security and Virus Scanning

```
@Component
public class FileSecurityService {

    private final List<String> allowedMimeTypes = Arrays.asList(
        "application/pdf",
        "image/jpeg", "image/png", "image/gif",
        "application/msword",

"application/vnd.openxmlformats-officedocument.wordprocessingml.document",
        "application/vnd.ms-excel",
        "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"
    );

    public void validateFile(MultipartFile file) {
        // File size validation
        if (file.getSize() > MAX_FILE_SIZE) {
            throw new ValidationException("File size exceeds maximum limit");
        }

        // MIME type validation
        if (!allowedMimeTypes.contains(file.getContentType())) {
            throw new ValidationException("File type not allowed");
        }

        // File signature validation
        if (!isValidFileSignature(file)) {
            throw new ValidationException("Invalid file signature");
        }

        // Virus scanning
        if (!scanForViruses(file)) {
            throw new SecurityException("File contains malicious content");
        }
    }

    private boolean isValidFileSignature(MultipartFile file) {
        // Implement file signature validation
        return true;
    }

    private boolean scanForViruses(MultipartFile file) {
        // Integrate with antivirus service (ClamAV, etc.)
        return true;
    }
}
```

Development Guidelines

Code Standards

Java Coding Standards

- **Java Version**: Java 17+ with modern language features
- **Code Style**: Google Java Style Guide
- **Naming Conventions**:
  - Classes: PascalCase
  - Methods/Variables: camelCase
  - Constants: UPPER_SNAKE_CASE
  - Packages: lowercase

Spring Boot Best Practices

```
// Service Layer Example
@Service
@Transactional(readOnly = true)
@Slf4j
public class DocumentService {

    private final DocumentRepository documentRepository;
    private final DocumentTypeService documentTypeService;
    private final FileStorageService fileStorageService;

    public DocumentService(DocumentRepository documentRepository,
                           DocumentTypeService documentTypeService,
                           FileStorageService fileStorageService) {
        this.documentRepository = documentRepository;
        this.documentTypeService = documentTypeService;
        this.fileStorageService = fileStorageService;
    }

    @Transactional
    public Document createDocument(CreateDocumentRequest request,
MultipartFile file) {
        // Validate document type access
        DocumentType documentType = documentTypeService
            .findByIdAndValidateAccess(request.getDocumentTypeId());

        // Store file
        FileMetadata fileMetadata = fileStorageService.store(file);

        // Create document entity
        Document document = Document.builder()
            .documentType(documentType)
            .fileName(file.getOriginalFilename())
            .fileSize(file.getSize())
            .storageKey(fileMetadata.getKey())
            .physicalLocation(request.getPhysicalLocation())
            .metadata(request.getMetadata())
            .build();

        return documentRepository.save(document);
    }
}
```

Error Handling

```
@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(EntityNotFoundException.class)
    public ResponseEntity<ErrorResponse>
handleEntityNotFound(EntityNotFoundException ex) {
        ErrorResponse error = ErrorResponse.builder()
            .code("ENTITY_NOT_FOUND")
            .message(ex.getMessage())
            .timestamp(Instant.now())
            .build();

        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(error);
    }

    @ExceptionHandler(ValidationException.class)
    public ResponseEntity<ErrorResponse> handleValidation(ValidationException
ex) {
        ErrorResponse error = ErrorResponse.builder()
            .code("VALIDATION_ERROR")
            .message(ex.getMessage())
            .errors(ex.getErrors())
            .timestamp(Instant.now())
            .build();

        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(error);
    }
}
```

Testing Strategy

Unit Testing

```
@ExtendWith(MockitoExtension.class)
class DocumentServiceTest {

    @Mock
    private DocumentRepository documentRepository;

    @Mock
    private FileStorageService fileStorageService;

    @InjectMocks
    private DocumentService documentService;

    @Test
    void createDocument_ShouldCreateDocument_WhenValidRequest() {
        // Given
        CreateDocumentRequest request = createValidRequest();
        MultipartFile file = createMockFile();

        when(fileStorageService.store(file))
            .thenReturn(new FileMetadata("storage-key", "url"));

        // When
        Document result = documentService.createDocument(request, file);
```

```
        // Then
        assertThat(result).isNotNull();

assertThat(result.getFileName()).isEqualTo(file.getOriginalFilename());
        verify(documentRepository).save(any(Document.class));
    }
}
```

Integration Testing

```
@SpringBootTest
@Testcontainers
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE)
class DocumentControllerIntegrationTest {

    @Container
    static PostgreSQLContainer<?> postgres = new
PostgreSQLContainer<>("postgres:14")
            .withDatabaseName("bika_test")
            .withUsername("test")
            .withPassword("test");

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    void createDocument_ShouldReturn201_WhenValidRequest() {
        // Test implementation
    }
}
```

Performance Guidelines

Database Optimization

- **Connection Pooling**: HikariCP with optimized settings
- **Query Optimization**: Use JPA Criteria API for complex queries
- **Indexing Strategy**: Proper indexing on search fields
- **Pagination**: Always use pagination for list endpoints

Caching Strategy

```
@Service
public class DocumentTypeService {

    @Cacheable(value = "documentTypes", key = "#id")
    public DocumentType findById(String id) {
        return documentTypeRepository.findById(id)
            .orElseThrow(() -> new EntityNotFoundException("Document type not
found"));
    }

    @CacheEvict(value = "documentTypes", key = "#result.id")
    public DocumentType update(DocumentType documentType) {
        return documentTypeRepository.save(documentType);
    }
}
```

File Handling

- **Streaming**: Use streaming for large file uploads/downloads
- **Validation**: File type and size validation
- **Virus Scanning**: Integration with antivirus services
- **Compression**: Automatic compression for eligible file types

Monitoring and Observability

Logging Configuration

```
logging:
  level:
    com.bika: INFO
    org.springframework.security: DEBUG
  pattern:
    console: "%d{HH:mm:ss.SSS} %-5level [%thread] %logger{36} - %msg%n"
    file: "%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level [%thread] %logger{36} -
%msg%n"
  file:
    name: logs/bika.log
    max-size: 100MB
    max-history: 30
```

Metrics and Health Checks

```
@Component
public class CustomHealthIndicator implements HealthIndicator {

    private final DocumentRepository documentRepository;

    @Override
    public Health health() {
        try {
            long count = documentRepository.count();
            return Health.up()
                .withDetail("totalDocuments", count)
                .build();
        } catch (Exception e) {
            return Health.down()
                .withDetail("error", e.getMessage())
                .build();
        }
    }
}
```

Development Guidelines (Extended)

API Development Standards

```
// Standard API Response Wrapper
@Data
@Builder
public class ApiResponse<T> {
    private boolean success;
    private T data;
    private String message;
```

```
    private List<String> errors;
    private Instant timestamp;
    private String requestId;

    public static <T> ApiResponse<T> success(T data) {
        return ApiResponse.<T>builder()
            .success(true)
            .data(data)
            .timestamp(Instant.now())
            .requestId(MDC.get("requestId"))
            .build();
    }

    public static <T> ApiResponse<T> error(String message, List<String>
errors) {
        return ApiResponse.<T>builder()
            .success(false)
            .message(message)
            .errors(errors)
            .timestamp(Instant.now())
            .requestId(MDC.get("requestId"))
            .build();
    }
}

// Base Controller with common functionality
@RestController
public abstract class BaseController {

    protected final Logger logger = LoggerFactory.getLogger(getClass());

    @Value("${app.api.version:v1}")
    protected String apiVersion;

    protected <T> ResponseEntity<ApiResponse<T>> success(T data) {
        return ResponseEntity.ok(ApiResponse.success(data));
    }

    protected <T> ResponseEntity<ApiResponse<T>> created(T data) {
        return ResponseEntity.status(HttpStatus.CREATED)
            .body(ApiResponse.success(data));
    }

    protected ResponseEntity<ApiResponse<Void>> noContent() {
        return ResponseEntity.noContent().build();
    }
}
```

Service Layer Patterns

```
// Generic Service Interface
public interface BaseService<T, ID> {
    T create(T entity);
    T update(ID id, T entity);
    T findById(ID id);
    Page<T> findAll(Pageable pageable);
    void delete(ID id);
}

// Abstract Service Implementation
@Transactional(readOnly = true)
public abstract class AbstractService<T, ID> implements BaseService<T, ID> {
```

```
    protected abstract JpaRepository<T, ID> getRepository();
    protected abstract String getEntityName();

    @Override
    public T findById(ID id) {
        return getRepository().findById(id)
            .orElseThrow(() -> new EntityNotFoundException(
                getEntityName() + " not found with id: " + id));
    }

    @Override
    @Transactional
    public T create(T entity) {
        validateEntity(entity);
        T saved = getRepository().save(entity);
        publishEvent(new EntityCreatedEvent<>(saved));
        return saved;
    }

    protected void validateEntity(T entity) {
        // Override in concrete implementations
    }

    protected void publishEvent(ApplicationEvent event) {
        ApplicationContextHolder.getApplicationContext().publishEvent(event);
    }
}
```

Caching Strategy Implementation

```
@Configuration
@EnableCaching
public class CacheConfig {

    @Bean
    public CacheManager cacheManager() {
        RedisCacheManager.Builder builder = RedisCacheManager
            .RedisCacheManagerBuilder
            .fromConnectionFactory(redisConnectionFactory())
            .cacheDefaults(cacheConfiguration());

        return builder.build();
    }

    private RedisCacheConfiguration cacheConfiguration() {
        return RedisCacheConfiguration.defaultCacheConfig()
            .entryTtl(Duration.ofMinutes(60))
            .serializeKeysWith(RedisSerializationContext.SerializationPair
                .fromSerializer(new StringRedisSerializer()))
            .serializeValuesWith(RedisSerializationContext.SerializationPair
                .fromSerializer(new GenericJackson2JsonRedisSerializer()));
    }
}

// Service with caching
@Service
@CacheConfig(cacheNames = "documentTypes")
public class DocumentTypeService extends AbstractService<DocumentType,
String> {

    @Override
```

```
    @Cacheable(key = "#id")
    public DocumentType findById(String id) {
        return super.findById(id);
    }

    @Override
    @CacheEvict(key = "#result.id")
    public DocumentType create(DocumentType documentType) {
        return super.create(documentType);
    }

    @CacheEvict(key = "#id")
    public DocumentType update(String id, DocumentType documentType) {
        return super.update(id, documentType);
    }
}
```

Database Design

Database Schema Overview

The database follows **Third Normal Form (3NF)** principles to ensure data integrity and minimize redundancy while supporting multi-tenancy and hierarchical data structures.

Core Entities

Companies Table

```
CREATE TABLE companies (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    phone VARCHAR(20),
    status VARCHAR(20) DEFAULT 'PENDING',
    storage_limit BIGINT DEFAULT 21474836480, -- 20GB default
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_by UUID,
    updated_by UUID
);
```

Users Table

```
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    company_id UUID NOT NULL REFERENCES companies(id),
    email VARCHAR(255) NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    phone VARCHAR(20),
    role VARCHAR(50) NOT NULL,
    status VARCHAR(20) DEFAULT 'ACTIVE',
    last_login TIMESTAMP,
    mfa_enabled BOOLEAN DEFAULT FALSE,
```

```
    mfa_secret VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(company_id, email)
);
```

Departments Table

```
CREATE TABLE departments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    company_id UUID NOT NULL REFERENCES companies(id),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    manager_id UUID REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(company_id, name)
);
```

Document Types Table

```
CREATE TABLE document_types (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    company_id UUID NOT NULL REFERENCES companies(id),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_by UUID REFERENCES users(id),
    UNIQUE(company_id, name)
);
```

Document Type Fields Table

```
CREATE TABLE document_type_fields (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    document_type_id UUID NOT NULL REFERENCES document_types(id),
    field_name VARCHAR(100) NOT NULL,
    field_label VARCHAR(255) NOT NULL,
    field_type VARCHAR(50) NOT NULL, -- TEXT, NUMBER, DATE, BOOLEAN, SELECT
    is_required BOOLEAN DEFAULT FALSE,
    validation_rules JSONB,
    default_value TEXT,
    field_order INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(document_type_id, field_name)
);
```

Extended Schema

```
-- Documents Table (Main entity for archived documents)
CREATE TABLE documents (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    company_id UUID NOT NULL REFERENCES companies(id),
    document_type_id UUID NOT NULL REFERENCES document_types(id),
    folder_id UUID REFERENCES folders(id),
    file_name VARCHAR(255) NOT NULL,
    file_size BIGINT NOT NULL,
    file_type VARCHAR(100),
```

```
    storage_key VARCHAR(500) NOT NULL, -- S3/MinIO key
    storage_url VARCHAR(1000),
    metadata JSONB DEFAULT '{}',
    physical_location JSONB NOT NULL,
    search_vector tsvector,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_by UUID NOT NULL REFERENCES users(id),
    updated_by UUID REFERENCES users(id)
);

-- Folders Table (Hierarchical folder structure)
CREATE TABLE folders (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    company_id UUID NOT NULL REFERENCES companies(id),
    parent_folder_id UUID REFERENCES folders(id),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    path VARCHAR(1000), -- Materialized path for quick lookups
    level INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_by UUID NOT NULL REFERENCES users(id),
    UNIQUE(company_id, parent_folder_id, name)
);

-- Personal Drive Files
CREATE TABLE drive_files (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id),
    folder_id UUID REFERENCES drive_folders(id),
    file_name VARCHAR(255) NOT NULL,
    original_name VARCHAR(255) NOT NULL,
    file_size BIGINT NOT NULL,
    file_type VARCHAR(100),
    storage_key VARCHAR(500) NOT NULL,
    description TEXT,
    is_shared BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Drive Folders
CREATE TABLE drive_folders (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id),
    parent_folder_id UUID REFERENCES drive_folders(id),
    name VARCHAR(255) NOT NULL,
    path VARCHAR(1000),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, parent_folder_id, name)
);

-- User-Department Associations
CREATE TABLE user_departments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id),
    department_id UUID NOT NULL REFERENCES departments(id),
    role VARCHAR(50) DEFAULT 'MEMBER', -- MEMBER, MANAGER
    assigned_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    assigned_by UUID REFERENCES users(id),
    UNIQUE(user_id, department_id)
```

```
);

-- Document Type Access Control
CREATE TABLE document_type_departments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    document_type_id UUID NOT NULL REFERENCES document_types(id),
    department_id UUID NOT NULL REFERENCES departments(id),
    access_level VARCHAR(20) DEFAULT 'READ_WRITE', -- READ_ONLY, READ_WRITE
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(document_type_id, department_id)
);

-- Audit Logs
CREATE TABLE audit_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    company_id UUID NOT NULL REFERENCES companies(id),
    user_id UUID REFERENCES users(id),
    entity_type VARCHAR(50) NOT NULL, -- DOCUMENT, USER, COMPANY, etc.
    entity_id UUID NOT NULL,
    action VARCHAR(50) NOT NULL, -- CREATE, UPDATE, DELETE, VIEW, DOWNLOAD
    old_values JSONB,
    new_values JSONB,
    ip_address INET,
    user_agent TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- System Settings
CREATE TABLE system_settings (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    company_id UUID REFERENCES companies(id), -- NULL for global settings
    setting_key VARCHAR(100) NOT NULL,
    setting_value JSONB NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(company_id, setting_key)
);
```

Data Relationships

```
erDiagram
    COMPANIES ||--o{ USERS : "has"
    COMPANIES ||--o{ DEPARTMENTS : "contains"
    COMPANIES ||--o{ DOCUMENT_TYPES : "defines"
    USERS ||--o{ DOCUMENTS : "uploads"
    USERS ||--o{ DRIVE_FILES : "owns"
    DEPARTMENTS ||--o{ USER_DEPARTMENTS : "includes"
    DOCUMENT_TYPES ||--o{ DOCUMENT_TYPE_FIELDS : "has"
    DOCUMENT_TYPES ||--o{ DOCUMENTS : "categorizes"
    FOLDERS ||--o{ DOCUMENTS : "contains"
    FOLDERS ||--o{ FOLDERS : "parent/child"
```

Indexing Strategy

Primary Indexes

```
-- Performance indexes for search operations
```

```
CREATE INDEX idx_documents_company_type ON documents(company_id,
document_type_id);
CREATE INDEX idx_documents_created_at ON documents(created_at DESC);
CREATE INDEX idx_documents_metadata_gin ON documents USING GIN(metadata);

-- Search optimization
CREATE INDEX idx_documents_search_text ON documents USING GIN(
    to_tsvector('english', file_name || ' ' || COALESCE(metadata::text, ''))
);

-- Physical location search
CREATE INDEX idx_documents_physical_location ON documents(
    (physical_location->>'office'),
    (physical_location->>'cupboard'),
    (physical_location->>'drawer')
);
```

Composite Indexes

```
-- User access patterns
CREATE INDEX idx_user_company_dept ON user_departments(user_id,
department_id);
CREATE INDEX idx_document_access ON documents(company_id, created_by,
created_at DESC);

-- Audit and reporting
CREATE INDEX idx_audit_logs_entity ON audit_logs(entity_type, entity_id,
created_at DESC);
```

Database Triggers and Functions

```
-- Update search vector when document is inserted/updated
CREATE OR REPLACE FUNCTION update_document_search_vector()
RETURNS TRIGGER AS $$
BEGIN
    NEW.search_vector := to_tsvector('english',
        NEW.file_name || ' ' ||
        COALESCE(NEW.metadata::text, '') || ' ' ||
        COALESCE(NEW.physical_location::text, '')
    );
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_update_document_search_vector
    BEFORE INSERT OR UPDATE ON documents
    FOR EACH ROW EXECUTE FUNCTION update_document_search_vector();

-- Update folder path when folder hierarchy changes
CREATE OR REPLACE FUNCTION update_folder_path()
RETURNS TRIGGER AS $$
DECLARE
    parent_path VARCHAR(1000);
BEGIN
    IF NEW.parent_folder_id IS NULL THEN
        NEW.path := NEW.name;
        NEW.level := 0;
    ELSE
        SELECT path, level INTO parent_path, NEW.level
        FROM folders WHERE id = NEW.parent_folder_id;
```

```
        NEW.path := parent_path || '/' || NEW.name;
        NEW.level := NEW.level + 1;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_update_folder_path
    BEFORE INSERT OR UPDATE ON folders
    FOR EACH ROW EXECUTE FUNCTION update_folder_path();
```

Frontend Architecture

React Application Structure

```
src/
├── components/           # Reusable UI components
│   ├── common/          # Generic components
│   ├── forms/           # Form-specific components
│   ├── layout/          # Layout components
│   └── ui/              # Base UI components
├── pages/               # Page components
│   ├── auth/
│   ├── dashboard/
│   ├── documents/
│   ├── drive/
│   └── admin/
├── hooks/               # Custom React hooks
├── services/            # API service layer
├── store/               # State management
│   ├── slices/          # Redux slices
│   └── api/             # RTK Query API definitions
├── utils/               # Utility functions
├── types/               # TypeScript type definitions
├── constants/           # Application constants
└── styles/              # Global styles
```

State Management with Redux Toolkit

```
// store/slices/authSlice.ts
import { createSlice, PayloadAction } from '@reduxjs/toolkit';

interface AuthState {
```

```
  user: User | null;
  token: string | null;
  isAuthenticated: boolean;
  loading: boolean;
  error: string | null;
}

const initialState: AuthState = {
  user: null,
  token: null,
  isAuthenticated: false,
  loading: false,
  error: null,
};

const authSlice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    loginStart: (state) => {
      state.loading = true;
      state.error = null;
    },
    loginSuccess: (state, action: PayloadAction<{ user: User; token: string
}>) => {
      state.user = action.payload.user;
      state.token = action.payload.token;
      state.isAuthenticated = true;
      state.loading = false;
      state.error = null;
    },
    loginFailure: (state, action: PayloadAction<string>) => {
      state.loading = false;
      state.error = action.payload;
      state.isAuthenticated = false;
    },
    logout: (state) => {
      state.user = null;
      state.token = null;
      state.isAuthenticated = false;
    },
  },
});

export const { loginStart, loginSuccess, loginFailure, logout } =
authSlice.actions;
export default authSlice.reducer;
```

API Service Layer with RTK Query

```
// services/api/documentsApi.ts
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';
import { RootState } from '../store';

export const documentsApi = createApi({
  reducerPath: 'documentsApi',
  baseQuery: fetchBaseQuery({
    baseUrl: '/api/v1/documents',
    prepareHeaders: (headers, { getState }) => {
      const token = (getState() as RootState).auth.token;
      if (token) {
        headers.set('authorization', `Bearer ${token}`);
```

```
      }
      return headers;
    },
  }),
  tagTypes: ['Document', 'DocumentType', 'Folder'],
  endpoints: (builder) => ({
    getDocuments: builder.query<PaginatedResponse<Document>, SearchParams>({
      query: (params) => ({
        url: '/search',
        params,
      }),
      providesTags: ['Document'],
    }),
    createDocument: builder.mutation<Document, CreateDocumentRequest>({
      query: (data) => ({
        url: '',
        method: 'POST',
        body: data,
      }),
      invalidatesTags: ['Document'],
    }),
    getDocumentTypes: builder.query<DocumentType[], string>({
      query: (companyId) => `/types?companyId=${companyId}`,
      providesTags: ['DocumentType'],
    }),
  }),
});

export const {
  useGetDocumentsQuery,
  useCreateDocumentMutation,
  useGetDocumentTypesQuery
} = documentsApi;
```

Component Architecture

```
// components/documents/DocumentUpload.tsx
import React, { useState } from 'react';
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { z } from 'zod';
import { useCreateDocumentMutation } from '../../services/api/documentsApi';

const documentSchema = z.object({
  documentTypeId: z.string().uuid(),
  folderId: z.string().uuid().optional(),
  metadata: z.record(z.any()),
  physicalLocation: z.object({
    office: z.string().min(1),
    cupboard: z.string().min(1),
    drawer: z.string().min(1),
    folderColor: z.string().min(1),
  }),
});

type DocumentFormData = z.infer<typeof documentSchema>;

const DocumentUpload: React.FC = () => {
  const [file, setFile] = useState<File | null>(null);
  const [createDocument, { isLoading }] = useCreateDocumentMutation();
```

```
  const { register, handleSubmit, formState: { errors } } =
useForm<DocumentFormData>({
    resolver: zodResolver(documentSchema),
  });

  const onSubmit = async (data: DocumentFormData) => {
    if (!file) return;

    const formData = new FormData();
    formData.append('file', file);
    formData.append('data', JSON.stringify(data));

    try {
      await createDocument(formData).unwrap();
      // Handle success
    } catch (error) {
      // Handle error
    }
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)} className="space-y-6">
      {/* Form implementation */}
    </form>
  );
};
```

Custom Hooks

```
// hooks/usePermissions.ts
import { useSelector } from 'react-redux';
import { RootState } from '../store';
import { Permission, Role } from '../types/auth';

export const usePermissions = () => {
  const user = useSelector((state: RootState) => state.auth.user);

  const hasPermission = (permission: Permission): boolean => {
    if (!user) return false;
    return user.permissions.includes(permission);
  };

  const hasRole = (role: Role): boolean => {
    if (!user) return false;
    return user.role === role;
  };

  const canAccessDocumentType = (documentTypeId: string): boolean => {
    if (!user) return false;
    // Check if user's departments have access to this document type
    return user.departments.some(dept =>
      dept.documentTypes.includes(documentTypeId)
    );
  };

  return {
    hasPermission,
    hasRole,
    canAccessDocumentType,
    user,
  };
};
```

Deployment & Scalability

Docker Configuration

Backend Dockerfile

```
FROM openjdk:17-jdk-slim as builder
WORKDIR /app
COPY gradle/ gradle/
COPY gradlew build.gradle settings.gradle ./
COPY src ./src
RUN ./gradlew build -x test

FROM openjdk:17-jre-slim
WORKDIR /app
COPY --from=builder /app/build/libs/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Frontend Dockerfile

```
FROM node:18-alpine as builder
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
```

Docker Compose for Development

```
version: '3.8'

services:
  postgres:
    image: postgres:14
    environment:
      POSTGRES_DB: bika_dev
      POSTGRES_USER: bika
      POSTGRES_PASSWORD: bika123
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    command: redis-server --appendonly yes
    volumes:
      - redis_data:/data

  elasticsearch:
```

```
    image: elasticsearch:8.8.0
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false
    ports:
      - "9200:9200"
    volumes:
      - elasticsearch_data:/usr/share/elasticsearch/data

  minio:
    image: minio/minio:latest
    command: server /data --console-address ":9001"
    ports:
      - "9000:9000"
      - "9001:9001"
    environment:
      MINIO_ROOT_USER: minioadmin
      MINIO_ROOT_PASSWORD: minioadmin
    volumes:
      - minio_data:/data

  backend:
    build: ./backend
    ports:
      - "8080:8080"
    environment:
      SPRING_PROFILES_ACTIVE: dev
      DATABASE_URL: jdbc:postgresql://postgres:5432/bika_dev
      REDIS_URL: redis://redis:6379
      ELASTICSEARCH_URL: http://elasticsearch:9200
      MINIO_URL: http://minio:9000
    depends_on:
      - postgres
      - redis
      - elasticsearch
      - minio

  frontend:
    build: ./frontend
    ports:
      - "3000:80"
    depends_on:
      - backend

volumes:
  postgres_data:
  redis_data:
  elasticsearch_data:
  minio_data:
```

Kubernetes Deployment

Backend Deployment

```
# backend-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bika-backend
  namespace: bika-prod
spec:
```

```
replicas: 3
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 1
    maxUnavailable: 0
selector:
  matchLabels:
    app: bika-backend
template:
  metadata:
    labels:
      app: bika-backend
      version: v1
  spec:
    affinity:
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - bika-backend
            topologyKey: kubernetes.io/hostname
    containers:
    - name: backend
      image: bika/backend:latest
      ports:
      - containerPort: 8080
      env:
      - name: SPRING_PROFILES_ACTIVE
        value: "prod"
      - name: DATABASE_URL
        valueFrom:
          secretKeyRef:
            name: bika-secrets
            key: database-url
      - name: JVM_OPTS
        value: "-Xmx512m -Xms256m -XX:+UseG1GC -XX:MaxGCPauseMillis=200"
      resources:
        requests:
          memory: "512Mi"
          cpu: "250m"
        limits:
          memory: "1Gi"
          cpu: "500m"
      livenessProbe:
        httpGet:
          path: /actuator/health
          port: 8080
        initialDelaySeconds: 60
        periodSeconds: 30
        timeoutSeconds: 5
        failureThreshold: 3
      readinessProbe:
        httpGet:
          path: /actuator/health/readiness
          port: 8080
```

```
          initialDelaySeconds: 30
          periodSeconds: 10
          timeoutSeconds: 5
          failureThreshold: 3

---
apiVersion: v1
kind: Service
metadata:
  name: bika-backend-service
  namespace: bika-prod
spec:
  selector:
    app: bika-backend
  ports:
  - port: 80
    targetPort: 8080
  type: ClusterIP

---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: bika-backend-pdb
  namespace: bika-prod
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: bika-backend
```

Frontend Deployment

```
# frontend-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bika-frontend
  namespace: bika-prod
spec:
  replicas: 2
  selector:
    matchLabels:
      app: bika-frontend
  template:
    metadata:
      labels:
        app: bika-frontend
    spec:
      containers:
      - name: frontend
        image: bika/frontend:latest
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "64Mi"
            cpu: "50m"
          limits:
            memory: "128Mi"
            cpu: "100m"
        livenessProbe:
```

```
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 30
          periodSeconds: 30
        readinessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 5

---
apiVersion: v1
kind: Service
metadata:
  name: bika-frontend-service
  namespace: bika-prod
spec:
  selector:
    app: bika-frontend
  ports:
  - port: 80
    targetPort: 80
  type: ClusterIP
```

Ingress Configuration

```
# ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: bika-ingress
  namespace: bika-prod
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt-prod
    nginx.ingress.kubernetes.io/rate-limit: "100"
    nginx.ingress.kubernetes.io/rate-limit-window: "1m"
spec:
  tls:
  - hosts:
    - api.bika.com
    - app.bika.com
    secretName: bika-tls
  rules:
  - host: api.bika.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: bika-backend-service
            port:
              number: 80
  - host: app.bika.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
```

```
          service:
            name: bika-frontend-service
            port:
              number: 80
```

Auto-scaling Configuration

Horizontal Pod Autoscaler

```
# hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: bika-backend-hpa
  namespace: bika-prod
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: bika-backend
  minReplicas: 3
  maxReplicas: 15
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
  behavior:
    scaleUp:
      stabilizationWindowSeconds: 60
      policies:
      - type: Percent
        value: 50
        periodSeconds: 60
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Percent
        value: 10
        periodSeconds: 60

---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: bika-frontend-hpa
  namespace: bika-prod
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: bika-frontend
  minReplicas: 2
```

```
  maxReplicas: 8
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 80
```

Vertical Pod Autoscaler

```
# vpa.yaml
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: bika-backend-vpa
  namespace: bika-prod
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: bika-backend
  updatePolicy:
    updateMode: "Auto"
  resourcePolicy:
    containerPolicies:
    - containerName: backend
      maxAllowed:
        cpu: 1
        memory: 2Gi
      minAllowed:
        cpu: 100m
        memory: 256Mi
```

Database High Availability

PostgreSQL Cluster

```
# postgres-cluster.yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: postgres-cluster
  namespace: bika-prod
spec:
  instances: 3
  primaryUpdateStrategy: unsupervised

  postgresql:
    parameters:
      max_connections: "200"
      shared_buffers: "256MB"
      effective_cache_size: "1GB"
      maintenance_work_mem: "64MB"
      checkpoint_completion_target: "0.9"
      wal_buffers: "16MB"
      default_statistics_target: "100"
      random_page_cost: "1.1"
      effective_io_concurrency: "200"

  bootstrap:
    initdb:
```

```
      database: bika_prod
      owner: bika
      secret:
        name: postgres-credentials

  storage:
    size: 100Gi
    storageClass: fast-ssd

  backup:
    retentionPolicy: "30d"
    barmanObjectStore:
      destinationPath: "s3://bika-backups/postgres"
      s3Credentials:
        accessKeyId:
          name: backup-credentials
          key: ACCESS_KEY_ID
        secretAccessKey:
          name: backup-credentials
          key: SECRET_ACCESS_KEY
      wal:
        retention: "5d"
      data:
        retention: "30d"

  monitoring:
    enabled: true
```

Redis Cluster

```
# redis-cluster.yaml
apiVersion: redis.redis.opstreelabs.in/v1beta1
kind: RedisCluster
metadata:
  name: redis-cluster
  namespace: bika-prod
spec:
  clusterSize: 6
  kubernetesConfig:
    image: redis:7-alpine
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 200m
        memory: 256Mi
  storage:
    volumeClaimTemplate:
      spec:
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: 10Gi
        storageClassName: fast-ssd
```

Multi-Environment Configuration

Staging Environment

```
# staging/backend-deployment.yaml
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: bika-backend
  namespace: bika-staging
spec:
  replicas: 2
  selector:
    matchLabels:
      app: bika-backend
  template:
    metadata:
      labels:
        app: bika-backend
        env: staging
    spec:
      containers:
      - name: backend
        image: bika/backend:staging
        ports:
        - containerPort: 8080
        env:
        - name: SPRING_PROFILES_ACTIVE
          value: "staging"
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: bika-staging-secrets
              key: database-url
        resources:
          requests:
            memory: "256Mi"
            cpu: "125m"
          limits:
            memory: "512Mi"
            cpu: "250m"
        livenessProbe:
          httpGet:
            path: /actuator/health
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 30
        readinessProbe:
          httpGet:
            path: /actuator/health/readiness
            port: 8080
          initialDelaySeconds: 15
          periodSeconds: 10
```

Blue-Green Deployment Strategy

```
# blue-green-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: bika-backend-active
  namespace: bika-prod
spec:
  selector:
    app: bika-backend
    version: blue  # Switch to 'green' during deployment
  ports:
  - port: 80
```

```
    targetPort: 8080
  type: ClusterIP

---
# Switch script for blue-green deployment
apiVersion: v1
kind: ConfigMap
metadata:
  name: blue-green-switch
  namespace: bika-prod
data:
  switch.sh: |
    #!/bin/bash
    CURRENT=$(kubectl get service bika-backend-active -o
jsonpath='{.spec.selector.version}')
    if [ "$CURRENT" = "blue" ]; then
      NEW="green"
    else
      NEW="blue"
    fi
    kubectl patch service bika-backend-active -p
'{"spec":{"selector":{"version":"'$NEW'"}}}'
    echo "Switched from $CURRENT to $NEW"
```

Performance Optimization

Complete Connection Pool Configuration

```
# application-prod.yml
spring:
  datasource:
    hikari:
      maximum-pool-size: 20
      minimum-idle: 5
      connection-timeout: 30000
      idle-timeout: 600000
      max-lifetime: 1800000
      leak-detection-threshold: 60000
      pool-name: BikaHikariCP
      auto-commit: false

  jpa:
    properties:
      hibernate:
        jdbc:
          batch_size: 25
        order_inserts: true
        order_updates: true
        batch_versioned_data: true
        connection:
          provider_disables_autocommit: true
        query:
          in_clause_parameter_padding: true

  redis:
    jedis:
      pool:
        max-active: 20
        max-idle: 10
        min-idle: 2
        max-wait: 30000ms
```

```
    timeout: 5000ms
```

JVM Optimization

```
# JVM tuning for containers
env:
- name: JVM_OPTS
  value: >-
    -Xmx512m
    -Xms256m
    -XX:+UseG1GC
    -XX:MaxGCPauseMillis=200
    -XX:+UnlockExperimentalVMOptions
    -XX:+UseCGroupMemoryLimitForHeap
    -XX:NewRatio=1
    -XX:+OptimizeStringConcat
    -XX:+UseStringDeduplication
    -Djava.security.egd=file:/dev/./urandom
```

Helm Charts Configuration

Chart.yaml

```
apiVersion: v2
name: bika
description: A Helm chart for Bika application
type: application
version: 1.0.0
appVersion: "1.0.0"
```

values.yaml

```
# Default values for bika
backend:
  replicaCount: 3
  image:
    repository: bika/backend
    tag: latest
    pullPolicy: IfNotPresent
  service:
    type: ClusterIP
    port: 80
  resources:
    requests:
      memory: "512Mi"
      cpu: "250m"
    limits:
      memory: "1Gi"
      cpu: "500m"
  autoscaling:
    enabled: true
    minReplicas: 3
    maxReplicas: 15
    targetCPUUtilizationPercentage: 70

frontend:
  replicaCount: 2
  image:
    repository: bika/frontend
    tag: latest
    pullPolicy: IfNotPresent
  service:
    type: ClusterIP
```

```
    port: 80
  resources:
    requests:
      memory: "64Mi"
      cpu: "50m"
    limits:
      memory: "128Mi"
      cpu: "100m"

ingress:
  enabled: true
  className: nginx
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-prod
  hosts:
    - host: api.bika.com
      paths:
        - path: /
          pathType: Prefix
          service: backend
    - host: app.bika.com
      paths:
        - path: /
          pathType: Prefix
          service: frontend
  tls:
    - secretName: bika-tls
      hosts:
        - api.bika.com
        - app.bika.com

postgresql:
  enabled: true
  global:
    postgresql:
      auth:
        username: bika
        database: bika_prod
  primary:
    persistence:
      enabled: true
      size: 100Gi
    resources:
      requests:
        memory: 256Mi
        cpu: 250m
      limits:
        memory: 512Mi
        cpu: 500m
```

CI/CD Pipeline Configuration

Enhanced GitHub Actions Workflow

```
# .github/workflows/deploy.yml
name: Build and Deploy

on:
  push:
    branches: [main, develop, staging]
  pull_request:
```

```yaml
    branches: [main]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  test:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:14
        env:
          POSTGRES_PASSWORD: postgres
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
      redis:
        image: redis:7-alpine
        options: >-
          --health-cmd "redis-cli ping"
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

    steps:
    - uses: actions/checkout@v4

    - name: Set up JDK 17
      uses: actions/setup-java@v4
      with:
        java-version: '17'
        distribution: 'temurin'
        cache: gradle

    - name: Run tests
      run: ./gradlew test

    - name: Run integration tests
      run: ./gradlew integrationTest

    - name: Generate test report
      uses: dorny/test-reporter@v1
      if: success() || failure()
      with:
        name: Test Results
        path: build/test-results/test/*.xml
        reporter: java-junit

    - name: Build application
      run: ./gradlew build

  build-and-push:
    needs: test
    runs-on: ubuntu-latest
    if: github.event_name == 'push'

    permissions:
      contents: read
```

```
      packages: write

    steps:
    - uses: actions/checkout@v4

    - name: Log in to Container Registry
      uses: docker/login-action@v3
      with:
        registry: ${{ env.REGISTRY }}
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}

    - name: Extract metadata
      id: meta
      uses: docker/metadata-action@v5
      with:
        images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
        tags: |
          type=ref,event=branch
          type=ref,event=pr
          type=sha,prefix={{branch}}-

    - name: Build and push Backend image
      uses: docker/build-push-action@v5
      with:
        context: ./backend
        push: true
        tags: ${{ steps.meta.outputs.tags }}-backend
        labels: ${{ steps.meta.outputs.labels }}

    - name: Build and push Frontend image
      uses: docker/build-push-action@v5
      with:
        context: ./frontend
        push: true
        tags: ${{ steps.meta.outputs.tags }}-frontend
        labels: ${{ steps.meta.outputs.labels }}

  deploy-staging:
    needs: build-and-push
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/develop'
    environment: staging

    steps:
    - uses: actions/checkout@v4

    - name: Configure kubectl
      uses: azure/k8s-set-context@v3
      with:
        method: kubeconfig
        kubeconfig: ${{ secrets.KUBE_CONFIG }}

    - name: Deploy to staging
      run: |
        helm upgrade --install bika-staging ./helm/bika \
          --namespace bika-staging \
          --create-namespace \
          --set backend.image.tag=${{ github.sha }}-backend \
          --set frontend.image.tag=${{ github.sha }}-frontend \
          --values ./helm/bika/values-staging.yaml
```

```
    - name: Run smoke tests
      run: |
        kubectl wait --for=condition=ready pod -l app=bika-backend -n
bika-staging --timeout=300s
        curl -f https://staging-api.bika.com/actuator/health || exit 1

  deploy-production:
    needs: [build-and-push, deploy-staging]
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    environment: production

    steps:
    - uses: actions/checkout@v4

    - name: Configure kubectl
      uses: azure/k8s-set-context@v3
      with:
        method: kubeconfig
        kubeconfig: ${{ secrets.KUBE_CONFIG }}

    - name: Deploy to production (Blue-Green)
      run: |
        # Get current active version
        CURRENT=$(kubectl get service bika-backend-active -n bika-prod -o
jsonpath='{.spec.selector.version}' || echo "blue")
        NEW=$([ "$CURRENT" = "blue" ] && echo "green" || echo "blue")

        # Deploy new version
        helm upgrade --install bika-$NEW ./helm/bika \
          --namespace bika-prod \
          --create-namespace \
          --set backend.image.tag=${{ github.sha }}-backend \
          --set frontend.image.tag=${{ github.sha }}-frontend \
          --set backend.version=$NEW \
          --values ./helm/bika/values-prod.yaml

        # Wait for deployment to be ready
        kubectl wait --for=condition=ready pod -l
app=bika-backend,version=$NEW -n bika-prod --timeout=600s

        # Run health checks
        kubectl port-forward svc/bika-backend-$NEW 8080:80 -n bika-prod &
        sleep 5
        curl -f http://localhost:8080/actuator/health || exit 1

        # Switch traffic to new version
        kubectl patch service bika-backend-active -n bika-prod -p
'{"spec":{"selector":{"version":"'$NEW'"}}}'

        echo "Successfully deployed version $NEW"

    - name: Cleanup old version
      run: |
        # Wait 5 minutes before cleanup
        sleep 300
        OLD=$([ "$NEW" = "blue" ] && echo "green" || echo "blue")
        helm uninstall bika-$OLD -n bika-prod || true
```

Monitoring and Observability

Application Configuration

```
# application-prod.yml
management:
  endpoints:
    web:
      exposure:
        include: health,info,metrics,prometheus
  endpoint:
    health:
      show-details: always
      probes:
        enabled: true
  metrics:
    export:
      prometheus:
        enabled: true
    distribution:
      percentiles-histogram:
        http.server.requests: true
  tracing:
    sampling:
      probability: 0.1

logging:
  level:
    com.bika: INFO
    org.springframework.security: WARN
    org.hibernate.SQL: WARN
  pattern:
    console: "%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level [%X{requestId}]
%logger{36} - %msg%n"
  appender:
    loki:
      url: http://loki:3100/loki/api/v1/push
```

ServiceMonitor for Prometheus

```
# servicemonitor.yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: bika-backend-metrics
  namespace: bika-prod
spec:
  selector:
    matchLabels:
      app: bika-backend
  endpoints:
  - port: http
    path: /actuator/prometheus
    interval: 30s
```

Storage Configuration

StorageClass for High Performance

```
# storage-class.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast-ssd
```

```
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp3
  iops: "3000"
  throughput: "125"
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
```

MinIO Distributed Setup

```
# minio-distributed.yaml
apiVersion: v1
kind: Service
metadata:
  name: minio-distributed
  namespace: bika-prod
spec:
  clusterIP: None
  selector:
    app: minio-distributed
  ports:
  - port: 9000
    name: minio

---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: minio-distributed
  namespace: bika-prod
spec:
  serviceName: minio-distributed
  replicas: 4
  selector:
    matchLabels:
      app: minio-distributed
  template:
    metadata:
      labels:
        app: minio-distributed
    spec:
      containers:
      - name: minio
        image: minio/minio:latest
        command:
        - /bin/bash
        - -c
        args:
        - minio server
http://minio-distributed-{0...3}.minio-distributed.bika-prod.svc.cluster.loca
l/data --console-address ":9001"
        ports:
        - containerPort: 9000
        - containerPort: 9001
        env:
        - name: MINIO_ROOT_USER
          valueFrom:
            secretKeyRef:
              name: minio-credentials
              key: username
        - name: MINIO_ROOT_PASSWORD
          valueFrom:
```

```
            secretKeyRef:
              name: minio-credentials
              key: password
        resources:
          requests:
            memory: 256Mi
            cpu: 100m
          limits:
            memory: 512Mi
            cpu: 200m
        volumeMounts:
        - name: data
          mountPath: /data
  volumeClaimTemplates:
  - metadata:
      name: data
    spec:
      accessModes: ["ReadWriteOnce"]
      storageClassName: fast-ssd
      resources:
        requests:
          storage: 100Gi
```

Network Policies

Security Network Policies

```
# network-policies.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: bika-backend-netpol
  namespace: bika-prod
spec:
  podSelector:
    matchLabels:
      app: bika-backend
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: ingress-nginx
    - podSelector:
        matchLabels:
          app: bika-frontend
    ports:
    - protocol: TCP
      port: 8080
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: postgres-cluster
    ports:
    - protocol: TCP
      port: 5432
  - to:
    - podSelector:
```

```
      matchLabels:
        app: redis-cluster
  ports:
  - protocol: TCP
    port: 6379
- to: []
  ports:
  - protocol: TCP
    port: 53
  - protocol: UDP
    port: 53
```