E-Repository System - Universitas Dumai

A comprehensive digital repository system for academic books and research papers, built with modern web technologies.

System Architecture

Backend (Go/Gin)

- **Framework**: Gin-Gonic for RESTful API
- **Database**: MySQL 8.0 with comprehensive schema
- **ORM**: GORM for database operations
- **Authentication**: JWT-based authentication
- **Security**: Bcrypt password hashing, role-based access control

Frontend (Next.js)

- **Framework**: Next.js 15.3.2 with App Router
- **Language**: TypeScript for type safety
- **Styling**: Tailwind CSS for modern UI
- **State Management**: React Context API
- **HTTP Client**: Axios with interceptors

Database Schema

- **Users**: Role-based user management (public, user, admin)
- **Content**: Books and papers with metadata
- **Categories**: Organized content classification
- **Relationships**: User-content interactions, author relationships
- **Activity Tracking**: User activity logs and download statistics

Project Structure

```
e-repository/
  backend/                    # Go API server
     cmd/main.go            # Application entry point
     configs/               # Configuration management
     internal/
        models/            # Data models and DTOs
        handlers/          # HTTP request handlers
        middleware/        # Authentication middleware
        database/          # Database connection and migrations
        utils/             # Utility functions
     go.mod                 # Go dependencies
     Dockerfile             # Backend container
  frontend/                  # Next.js application
     src/
```

```
        app/                # App Router pages
        components/         # Reusable components
        contexts/           # React Context providers
        lib/                # API client and utilities
    package.json            # Node.js dependencies
    Dockerfile              # Frontend container
database_schema.sql         # Database schema
sample_data.sql             # Comprehensive sample data
simple_sample_data.sql      # Basic sample data
docker-compose.yml          # Docker services configuration
```

Quick Start

Prerequisites

- Docker and Docker Compose
- Git

Installation

1. **Clone the repository**

   ```
   git clone <repository-url>
   cd e-repository
   ```

2. **Start the services**

   ```
   docker-compose up -d
   ```

3. **Verify the services**

   ```
   # Check all services are running
   docker-compose ps

   # Test API health
   curl http://localhost:8080/api/v1/health

   # Access frontend
   open http://localhost:3000
   ```

Demo Accounts

The system comes with pre-configured demo accounts for testing:

Role	Email	Password	Description
Admin	admin@demo.com	password123	Full administrative access
User	user@demo.com	password123	Standard user privileges
User	john.smith@demo.com	password123	Sample student account

Role	Email	Password	Description
User	sarah.johnson@demo.com	password123	Sample student account

Demo Data Included

- **5 Books**: Computer Science, Mathematics, and Physics textbooks
- **4 Papers**: Research papers in various academic fields
- **4 Categories**: Computer Science, Mathematics, Physics, Engineering
- **Activity Logs**: Sample user interactions
- **Relationships**: Author associations and category mappings

API Endpoints

Public Endpoints

- `GET /api/v1/health` - Health check
- `POST /api/v1/auth/login` - User authentication
- `POST /api/v1/auth/register` - User registration
- `GET /api/v1/books` - List books (with search & pagination)
- `GET /api/v1/papers` - List papers (with search & pagination)

Protected Endpoints (Requires Authentication)

- `GET /api/v1/profile` - User profile
- `GET /api/v1/books/:id/download` - Download book
- `GET /api/v1/papers/:id/download` - Download paper

Admin Endpoints (Requires Admin Role)

- `POST /api/v1/admin/books` - Create book
- `PUT /api/v1/admin/books/:id` - Update book
- `DELETE /api/v1/admin/books/:id` - Delete book
- `POST /api/v1/admin/papers` - Create paper
- `PUT /api/v1/admin/papers/:id` - Update paper
- `DELETE /api/v1/admin/papers/:id` - Delete paper
- `GET /api/v1/admin/stats` - System statistics

Configuration

Environment Variables

Backend (.env)

```
DB_HOST=mysql
DB_PORT=3306
DB_NAME=test_db2
DB_USER=e_repositori
```

```
DB_PASSWORD=secure_password_here
JWT_SECRET=your_jwt_secret_key_here
PORT=8080
```

Frontend

```
NEXT_PUBLIC_API_URL=http://localhost:8080
```

Docker Services

- **MySQL**: Port 3307 (mapped from 3306)
- **API Server**: Port 8080
- **Frontend**: Port 3000

Features

Implemented Features

- User authentication and authorization
- Role-based access control (public, user, admin)
- Book and paper management
- Search and pagination
- Category organization
- File upload handling
- Activity logging
- Download tracking
- Responsive web interface
- REST API with comprehensive endpoints

Future Enhancements

- File upload functionality
- Advanced search filters
- User dashboard with statistics
- Admin management panel
- Email notifications
- Advanced user management
- Content recommendation system
- API documentation (Swagger)

Development

Running in Development Mode

1. **Backend Development**

   ```
   cd backend
   go run cmd/main.go
   ```

2. **Frontend Development**

   ```
   cd frontend
   npm run dev
   ```

3. **Database Management**

   ```
   # Access MySQL
   docker exec -it e-repository-mysql mysql -u root -prootpassword test_db2

   # Load sample data
   docker exec -i e-repository-mysql mysql -u root -prootpassword < simple_sample_data.sql
   ```

Building for Production

```
# Build all services
docker-compose build

# Deploy to production
docker-compose up -d --scale frontend=2
```

Database Schema

The system uses a comprehensive MySQL schema with the following key tables:

- `users` - User accounts and profiles
- `books` - Book metadata and information
- `papers` - Research paper details
- `categories` - Content categorization
- `book_categories`, `paper_categories` - Category relationships
- `book_authors`, `paper_authors` - Author relationships
- `user_books`, `user_papers` - User content interactions
- `activity_logs` - User activity tracking
- `downloads` - Download history
- `file_uploads` - File management

Security Features

- JWT token-based authentication
- Bcrypt password hashing
- Role-based authorization
- CORS protection
- SQL injection prevention (GORM)
- Input validation and sanitization

Frontend Features

- **Modern UI**: Built with Tailwind CSS

- **Responsive Design**: Mobile-first approach
- **TypeScript**: Type-safe development
- **Search Functionality**: Real-time search
- **Pagination**: Efficient data loading
- **Authentication Flow**: Login/logout/registration
- **Error Handling**: User-friendly error messages
- **Loading States**: Better UX with loading indicators

Testing

API Testing

```
# Test authentication
curl -X POST http://localhost:8080/api/v1/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"admin@demo.com","password":"password123"}'

# Test book listing
curl http://localhost:8080/api/v1/books

# Test paper search
curl "http://localhost:8080/api/v1/papers?query=machine%20learning"
```

Frontend Testing

1. Navigate to http://localhost:3000
2. Use demo credentials to login
3. Browse books and papers
4. Test search functionality
5. Verify responsive design

Support

For questions or issues: - Check the logs: `docker-compose logs [service-name]` - Verify all services are running: `docker-compose ps` - Restart services: `docker-compose restart [service-name]`

Demo Access

Visit the application at: **http://localhost:3000**

Use the demo credentials provided in the login page to explore the system features.

Built with for Universitas Dumai