AJARLY

Project Overview

Ajarly is a comprehensive property rental platform built for the Egyptian market, connecting property owners with renters. The platform supports both short-term vacation rentals and long-term leases, featuring a robust booking system, payment integration, and multi-role dashboards.

Key Objectives

- Streamline property listing and discovery
- Facilitate secure bookings and payments
- Provide analytics for property owners
- Enable efficient property management
- Support bilingual (Arabic/English) user experience

Target Users

- Renters: Find and book properties
- Landlords: List and manage properties
- Brokers: Manage multiple properties for clients
- Admins: Platform oversight and moderation

Architecture

System Architecture

```
┌─────────────────────────────────────────────────────────────
─────────────────────────────────────────────────┐
│                 Frontend (React)                │
│   ┌──────────────────────────┐   ┌──────────────────────────┐
┌──────────────────────────┐   │
│   │   Pages      │   │  Components   │   │   Contexts    │      │
│   │  - Home       │   │  - Navbar       │   │  - Favorites │      │
│   │  - Search     │   │  - Cards       │   │  - Auth       │      │
│   │  - Details    │   │  - Forms       │   │               │      │
│   │  - Dashboard  │   │  - Modals      │   │               │      │
│   └──────────────────────────┘   └──────────────────────────┘
└──────────────────────────┘        │
└──────────────────────────────────────────────────────────────────────────
──────────────────────────────────────┘
                  │
                  │  REST API (HTTPS)
                  ▼
┌──────────────────────────────────────────────────────────────────────────
──────────────────────────────────────┐
│              Backend (Spring Boot)                    │
│   ┌──────────────────────────┐   ┌──────────────────────────┐
┌──────────────────────────┐        │
│   │ Controllers  │   │   Services   │   │ Repositories │     │
│   │ - Auth       │   │ - Property   │   │ - JPA/JDBC   │     │
│   │ - Property   │   │  - Booking   │   │              │     │
│   │ - Booking    │   │  - Payment   │   │              │     │
│   │ - Review     │   │ - Analytics  │   │              │     │
│   └──────────────────────────┘   └──────────────────────────┘
└──────────────────────────┘        │
│                                       │
│   ┌──────────────────────────┐   ┌──────────────────────────┐
┌──────────────────────────┐        │
│   │  Security    │   │ Scheduling   │   │   External   │     │
│   │  - JWT Auth  │   │   - Jobs     │   │ - Cloudinary │     │
│   │  - CORS      │   │ - Cron       │   │ - Fawry      │     │
│   └──────────────────────────┘   └──────────────────────────┘
└──────────────────────────┘        │
└──────────────────────────────────────────────────────────────────────────
──────────────────────────────────────┘
                  │
                  ▼
┌──────────────────────────────────────────────────────────────────────────
──────────────────────────────────────┐
│               Database (MySQL)                       │
│  - Users        - Bookings      - Analytics          │
│  - Properties    - Reviews       - Transactions        │
│  - Images        - Favorites     - Reports            │
└──────────────────────────────────────────────────────────────────────────
──────────────────────────────────────┘
                  │
                  ▼
┌──────────────────────────────────────────────────────────────────────────
──────────────────────────────────────┐
│              Cloud Services                        │
│  - Cloudinary (Image Storage)                          │
│  - Fawry (Payment Gateway - Simulated)                     │
```

Design Patterns Used

1. MVC Pattern: Controllers handle HTTP, Services contain business logic, Repositories manage data
2. DTO Pattern: Data Transfer Objects for API communication
3. Repository Pattern: Abstracted data access layer
4. Service Layer Pattern: Business logic separated from controllers
5. Factory Pattern: JWT token generation and validation
6. Observer Pattern: Event-driven scheduling tasks

Technology Stack

Backend

- Framework: Spring Boot 3.x
- Language: Java 17
- Build Tool: Maven
- Database: MySQL 8.0
- ORM: Spring Data JPA (Hibernate)
- Security: Spring Security + JWT
- Validation: Jakarta Validation
- Scheduling: Spring Scheduler
- API Documentation: (Recommended: Swagger/OpenAPI)

Frontend

- Framework: React 18.3.1
- Language: TypeScript
- Build Tool: Vite 6.3.5
- Routing: React Router DOM v7
- UI Components: Radix UI + shadcn/ui
- Styling: Tailwind CSS
- State Management: React Context API
- HTTP Client: Axios
- Forms: React Hook Form
- Notifications: Sonner

Cloud Services

- Image Storage: Cloudinary
- Hosting: Railway
- Payment Gateway: Fawry (Integration ready, currently simulated)

DevOps

- Version Control: Git
- CI/CD: Railway auto-deployment
- Environment Management: .env files
- Monitoring: Application logs

Features

1. Authentication & User Management

- Registration: Multi-role (Renter, Landlord, Broker, Admin)
- Login: JWT-based authentication
- Password Reset: Email-based recovery
- Profile Management: Avatar upload, bio, location
- Phone Verification: SMS verification system (ready for integration)

2. Property Management

- Property Listing: Create with images, amenities, pricing
- Image Upload: Multiple images with Cloudinary integration
- Property Types: Apartment, Villa, Chalet, Studio, etc.
- Rental Types: Vacation, Long-term, Both
- Status Management: Draft, Pending Approval, Active, Suspended
- Soft Delete: Properties can be recovered

3. Search & Discovery

- Advanced Search: Location, dates, guests, price range
- Filters: Bedrooms, bathrooms, amenities, property type
- Popular Locations: Trending destinations with statistics
- Categories: Browse by property type
- Autocomplete: Location suggestions

4. Booking System

- Availability Check: Real-time availability validation
- Booking Request: Renters submit booking requests

- Owner Confirmation: Owners approve/reject within 48 hours
- Auto-Expiry: Pending bookings expire after 48 hours
- Cancellation: Both parties can cancel with fee calculation
- Status Tracking: Pending, Confirmed, Completed, Cancelled

5. Payment Processing

- Payment Intent: Create payment for bookings
- Multiple Methods: Credit Card, Fawry, Vodafone Cash, Cash
- Payment Confirmation: Manual and webhook-based
- Refund Processing: Automated refund calculations
- Transaction History: Complete payment records
- Platform Fees: 10% platform fee, automatic calculation

6. Review System

- Multi-Criteria Ratings: Cleanliness, Accuracy, Communication, Location, Value
- Text Reviews: Title, detailed review, pros/cons
- Owner Responses: Owners can reply to reviews
- Auto-Approval: Reviews automatically approved (admin can moderate)
- Rating Aggregation: Automatic property rating updates

7. Favorites/Wishlist

- Add to Favorites: Save properties for later
- Notes: Add personal notes to favorites
- Quick Access: View all favorites in dashboard
- Sync: Real-time synchronization across devices

8. Analytics & Reporting

- Owner Dashboard: Revenue, bookings, views, ratings
- Property Analytics: Performance metrics per property
- Platform Analytics: Admin view of entire platform (Admin only)
- Daily Calculations: Scheduled analytics updates
- Charts & Graphs: Visual data representation

9. Admin Features

- Property Approval: Review and approve new listings
- User Management: Ban/unban users, verify IDs
- Report Moderation: Handle user reports
- Review Management: Approve/reject reviews

- Dashboard Statistics: Platform overview
- Audit Logs: Track admin actions

10. Subscription Plans

- Free Plan: 3 properties max
- Basic Plan: 10 properties, verified badge
- Professional Plan: Unlimited properties, analytics
- Enterprise Plan: All features + custom branding
- Rental Duration Plans: 1-day, 3-day specialized plans

API Documentation

Base URL

```
Production: https://ajarly-backend-production.up.railway.app
Development: http://localhost:8080
```

Authentication

All protected endpoints require JWT token in Authorization header:

```
Authorization: Bearer <token>
```

Core Endpoints

Authentication

```
POST   /api/v1/auth/register       - Register new user
POST   /api/v1/auth/login          - Login user
```

Properties

```
GET    /api/v1/properties          - Search properties (public)
GET    /api/v1/properties/{id}     - Get property details (public)
POST   /api/v1/properties          - Create property (auth)
PUT    /api/v1/properties/{id}     - Update property (owner)
DELETE /api/v1/properties/{id}     - Delete property (owner)
GET    /api/v1/properties/my-properties - Get user's properties (owner)
```

Property Images

```
POST   /api/v1/properties/{id}/images      - Upload images (owner)
GET    /api/v1/properties/{id}/images      - Get images (public)
```

DELETE /api/v1/properties/images/{imageId} - Delete image (owner)

PUT /api/v1/properties/images/{imageId}/cover - Set cover image (owner)

Bookings

```
POST   /api/v1/bookings           - Create booking (renter)
GET    /api/v1/bookings           - Get user bookings (renter)
GET    /api/v1/bookings/owner      - Get owner bookings (owner)
GET    /api/v1/bookings/{id}       - Get booking details (auth)
PUT    /api/v1/bookings/{id}/confirm - Confirm booking (owner)
PUT    /api/v1/bookings/{id}/reject  - Reject booking (owner)
PUT    /api/v1/bookings/{id}/cancel  - Cancel booking (both)
GET    /api/v1/bookings/availability/check - Check availability (public)
```

Reviews

```
POST   /api/v1/reviews            - Create review (renter)
GET    /api/v1/reviews/property/{id} - Get property reviews (public)
GET    /api/v1/reviews/my-reviews  - Get user's reviews (renter)
PUT    /api/v1/reviews/{id}/response - Owner response (owner)
PUT    /api/v1/reviews/{id}/approve  - Approve review (admin)
PUT    /api/v1/reviews/{id}/reject   - Reject review (admin)
GET    /api/v1/reviews/admin/all   - Get all reviews (admin)
```

Favorites

```
POST   /api/v1/favorites          - Add favorite (auth)
GET    /api/v1/favorites          - Get favorites (auth)
DELETE /api/v1/favorites/{propertyId} - Remove favorite (auth)
GET    /api/v1/favorites/check/{propertyId} - Check if favorited (auth)
```

Payments

```
POST   /api/v1/payments/create     - Create payment intent
POST   /api/v1/payments/confirm    - Confirm payment
POST   /api/v1/payments/refund     - Process refund
GET    /api/v1/payments/history    - Payment history
```

Analytics

```
GET    /api/v1/analytics/property/{id} - Property performance (owner)
GET    /api/v1/analytics/owner/dashboard - Owner dashboard (owner)
GET    /api/v1/analytics/admin/platform - Platform analytics (admin)
POST   /api/v1/analytics/admin/recalculate-ratings - Fix ratings (admin)
```

Search & Locations

```
POST   /api/v1/search              - Advanced search
GET    /api/v1/locations/suggestions - Location autocomplete
GET    /api/v1/locations/popular   - Popular locations
GET    /api/v1/locations/governorates - Get governorates
GET    /api/v1/locations/cities    - Get cities by governorate
```

User Profile

```
GET    /api/v1/users/profile       - Get profile (auth)
PUT    /api/v1/users/profile       - Update profile (auth)
PUT    /api/v1/users/password      - Change password (auth)
POST   /api/v1/users/upload-avatar - Upload avatar (auth)
POST   /api/v1/users/verify-phone  - Request phone verification (auth)
POST   /api/v1/users/verify-phone/confirm - Confirm verification code (auth)
```

Admin

```
GET    /api/v1/admin/dashboard     - Dashboard stats (admin)
GET    /api/v1/admin/properties/pending - Pending properties (admin)
PUT    /api/v1/admin/properties/{id}/approve - Approve property (admin)
PUT    /api/v1/admin/properties/{id}/reject  - Reject property (admin)
GET    /api/v1/admin/users         - Get all users (admin)
PUT    /api/v1/admin/users/{id}/ban - Ban user (admin)
PUT    /api/v1/admin/users/{id}/unban - Unban user (admin)
```

Response Format

All API responses follow this structure:

json
```json
{
  "success": true,
  "message": "Operation successful",
  "data": { /* response data */ },
  "timestamp": "2025-01-15T10:30:00"
}
```

Error responses:

json
```json
{
  "success": false,
  "message": "Error description",
  "errors": { /* validation errors */ },
  "timestamp": "2025-01-15T10:30:00"
```

}

Database Schema

Core Tables

users

sql
```
- user_id (PK)
- email (unique)
- password_hash
- phone_number (unique)
- phone_verified
- user_type (renter/landlord/broker/admin)
- first_name, last_name
- profile_photo
- bio
- governorate, city
- is_active
- national_id_verified

- created_at, updated_at
```

properties

sql
```
- property_id (PK)
- owner_id (FK -> users)
- title_ar, title_en
- description_ar, description_en
- slug (unique)
- property_type (apartment/villa/etc)
- rental_type (vacation/long_term/both)
- governorate, city, neighborhood
- latitude, longitude
- bedrooms, bathrooms, guests_capacity
- area_sqm
- furnished, pets_allowed, smoking_allowed
- price_per_night, price_per_month
- cleaning_fee, security_deposit
- status (draft/pending/active/suspended)
- view_count
- average_rating, total_reviews
- is_featured
```

- deleted, deleted_at, deleted_by
- created_at, updated_at

property_images

sql
- image_id (PK)
- property_id (FK -> properties)
- image_url
- thumbnail_url, medium_url, large_url
- image_order
- is_cover
- file_size, width, height
- uploaded_at

bookings

sql
- booking_id (PK)
- booking_reference (unique)
- property_id (FK -> properties)
- renter_id (FK -> users)
- owner_id (FK -> users)
- check_in_date, check_out_date
- number_of_nights, number_of_guests
- price_per_night, subtotal
- cleaning_fee, service_fee, total_price
- status (pending/confirmed/cancelled/completed)
- payment_status (unpaid/paid/refunded)
- requested_at, confirmed_at, expires_at

reviews

sql
- review_id (PK)
- booking_id (FK -> bookings)
- property_id (FK -> properties)
- reviewer_id (FK -> users)
- reviewee_id (FK -> users)
- overall_rating
- cleanliness_rating, accuracy_rating, etc.
- review_title, review_text
- pros, cons
- owner_response, owner_response_date
- is_approved
- created_at, updated_at

favorites
sql
- favorite_id (PK)
- user_id (FK -> users)
- property_id (FK -> properties)
- notes

- created_at

transactions
sql
- transaction_id (PK)
- transaction_reference (unique)
- user_id (FK -> users)
- booking_id (FK -> bookings)
- transaction_type
- amount, currency
- payment_method
- gateway_transaction_id
- status (pending/completed/failed/refunded)
- platform_fee_amount, owner_payout_amount

- created_at, completed_at

property_performance_analytics
sql
- performance_id (PK)
- property_id (FK -> properties)
- analytics_date
- total_views, unique_views
- booking_requests, booking_confirmations
- revenue, new_reviews

- created_at

reports
sql
- report_id (PK)
- reporter_id (FK -> users)
- report_type (property/user/review/message)
- reported_property_id, reported_user_id, etc.
- reason, description
- status (pending/investigating/resolved)
- priority (low/medium/high/urgent)
- assigned_to (FK -> users)
- resolved_by (FK -> users)

- action_taken
- created_at, resolved_at
```

### Relationships
- One-to-Many: User -> Properties, User -> Bookings, Property -> Images
- One-to-One: Booking -> Review
- Many-to-Many: Users <-> Properties (through Favorites)

---

## Authentication & Authorization

### JWT Implementation
- **Token Generation**: On successful login/registration
- **Token Content**: userId, email, role
- **Expiration**: 7 days (configurable)
- **Storage**: LocalStorage on client, validated on every request

### Security Features
- **Password Encryption**: BCrypt with salt
- **CORS Configuration**: Configured for specific origins
- **CSRF Protection**: Disabled (JWT-based stateless auth)
- **SQL Injection Protection**: JPA parameterized queries
- **XSS Protection**: Input validation and sanitization

### Role-Based Access Control (RBAC)
```
Renter:
- Browse properties
- Create bookings
- Write reviews
- Manage favorites

Landlord:
- All Renter permissions
- Create/manage properties
- Approve/reject bookings
- Respond to reviews
- View analytics

Broker:
- All Landlord permissions
- Manage multiple properties for clients

Admin:

- Full platform access
- User management
- Property approval/rejection
- Review moderation
- Platform analytics
- Report management

Endpoint Security Matrix

Endpoint Pattern	Public	Renter	Landlord	Broker	Admin
/api/v1/auth/**	✅	✅	✅	✅	✅
/api/v1/properties (GET)	✅	✅	✅	✅	✅
/api/v1/properties (POST)	❌	✅	✅	✅	✅
/api/v1/bookings/**	❌	✅	✅	✅	✅
/api/v1/admin/**	❌	❌	❌	❌	✅
/api/v1/analytics/admin/**	❌	❌	❌	❌	✅

Deployment Guide

Prerequisites

- Java 17+
- Node.js 18+
- MySQL 8.0+
- Cloudinary account
- Railway account (or alternative hosting)

Backend Deployment (Railway)

1. Environment Variables:

bash
```
# Database
MYSQLHOST=your-railway-host
MYSQLPORT=3306
```

```
MYSQLDATABASE=railway
MYSQLUSER=root
MYSQLPASSWORD=your-password

# JWT
JWT_SECRET=your-secret-key
JWT_EXPIRATION=604800000

# Cloudinary
CLOUDINARY_CLOUD_NAME=your-cloud-name
CLOUDINARY_API_KEY=your-api-key
CLOUDINARY_API_SECRET=your-api-secret

# Payment (Optional)
FAWRY_ENABLED=false
```

```
FAWRY_MERCHANT_CODE=test-code
```

2. Build Command:

bash

```
mvn clean package -DskipTests
```

3. Start Command:

bash

```
java -jar target/ajarly-backend-0.0.1-SNAPSHOT.jar
```

Frontend Deployment

1. Environment Variables:

bash

```
VITE_API_BASE_URL=https://your-backend-url.railway.app/api/v1
```

2. Build Command:

bash

```
npm run build
```

3. Deploy to Vercel/Netlify:

bash

```
# Vercel
```

```
vercel --prod

# Netlify
netlify deploy --prod
```

Database Migration

sql
```
-- Run schema.sql first
-- Then run subscription_data.sql
-- Finally, run any pending migrations
```

Development Setup

Backend Setup

1. Clone Repository:

bash
```
git clone https://github.com/yourusername/ajarly.git
cd ajarly/Backend
```

2. Configure Database:

properties
```
# src/main/resources/application.properties
spring.datasource.url=jdbc:mysql://localhost:3306/ajarly
spring.datasource.username=root
spring.datasource.password=yourpassword
```

3. Install Dependencies:

bash
```
mvn clean install
```

4. Run Application:

bash
```
mvn spring-boot:run
```

Backend will start on http://localhost:8080

Frontend Setup

1. Navigate to Frontend:

bash
```
cd ../Frontend
```

2. Install Dependencies:

bash
```
npm install
```

3. Configure API URL:

bash
```
# Create .env file
echo "VITE_API_BASE_URL=http://localhost:8080/api/v1" > .env
```

4. Run Development Server:

bash
```
npm run dev
```

Frontend will start on http://localhost:3355

Development Tools

- API Testing: Postman/Insomnia
- Database Client: MySQL Workbench/DBeaver
- Code Editor: VSCode/IntelliJ IDEA
- Version Control: Git

Testing

Backend Testing

bash
```
# Run all tests
mvn test
```

```
# Run specific test class
mvn -Dtest=PropertyServiceTest test

# With coverage
mvn test jacoco:report
```

Frontend Testing

```bash
# Run all tests
npm test

# Run in watch mode
npm run test:watch

# With coverage
npm run test:coverage
```

API Testing

```bash
# Import Postman collection
# Test all endpoints with various scenarios
```

Contributing

Code Style

- Backend: Follow Google Java Style Guide
- Frontend: Prettier + ESLint configuration
- Commits: Conventional Commits format

Pull Request Process

1. Fork the repository
2. Create feature branch (`git checkout -b feature/AmazingFeature`)
3. Commit changes (`git commit -m 'feat: Add amazing feature'`)
4. Push to branch (`git push origin feature/AmazingFeature`)
5. Open Pull Request

Branch Strategy

- main: Production-ready code
- develop: Development branch
- feature/*: New features
- bugfix/*: Bug fixes
- hotfix/*: Urgent production fixes

Known Issues & Roadmap

Known Issues

- Fawry payment integration is simulated (integration ready)
- SMS verification not connected to provider
- Email notifications not implemented
- Real-time chat system pending

Upcoming Features

- Real-time messaging between renters and owners
- Advanced calendar availability management
- Multi-currency support
- Mobile app (React Native)
- AI-powered property recommendations
- Virtual property tours (3D/VR)

Support & Contact

- Documentation: GitHub Wiki
- Issues: GitHub Issues
- Email: support@ajarly.com
- Discord: Join our community

License